Information Society Technologies

Project no. IST-033576

XtreemOS

Integrated Project
BUILDING AND PROMOTING A LINUX-BASED OPERATING SYSTEM TO SUPPORT VIRTUAL ORGANIZATIONS FOR NEXT GENERATION GRIDS

Linux XOS Specification
D2.1.1

Due date of deliverable: November 30^{th}, 2006
Actual submission date: January 11^{th}, 2007

Start date of project: June 1^{st} 2006

Type: Deliverable
WP number: WP2.1
Task number: T2.1.1

Responsible institution: INRIA
Editor & and editor's address: Christine Morin
IRISA/INRIA
Campus de Beaulieu
35042 RENNES Cedex
France

Version 1.0 / Last edited by Christine Morin / January 11^{th}, 2007

Project co-funded by the European Commission within the Sixth Framework Programme		
Dissemination Level		
PU	Public	$\surd$
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Revision history:

Version	Date	Authors	Institution	Section affected, comments
0.1	16/10/06	David Margery	INRIA	Initial template
0.2	23/10/06	Christine Morin	INRIA	Initial outline
0.3	24/10/06	Christine Morin	INRIA	Revised outline and responsabilities for editing the sections
0.4	25/10/06	Christine Morin	INRIA	Early draft of the introduction
0.5	30/10/06	David Margery	INRIA	First draft on checkpointing section
0.5.1	30/10/06	David Margery	INRIA	Added Thomas Ropars' state of the art paragraphs from the wiki
0.5.2	30/10/06	Luis Pablo Prieto	TID	Added VO management requirements
0.6.0	31/10/06	Haiyan Yu	ICT	Early draft of state of the art of VO
0.6.1	31/10/06	An Qin	ICT	User identity management, Resource Access Control management for VO
0.6.2	1/11/06	An Qin	ICT	Modification to Linux kernel
0.6.3	2/11/06	Erich Focht	NEC	Modification to Linux update
0.6.4	7/11/06	An Qin	ICT	Update Resources section, session definition, VO support, PAM plugins, API, account mapping
0.6.5	7/11/06	Yvon Jégou	INRIA	Update section Virtual Organizations in XtreemOS
0.7	7/11/06	Adrien Lebre	INRIA	Change the structure of the document to a latex report format
0.8	9/11/06	Yvon Jégou	INRIA	Virtual organization support update
0.9	14/11/06	Erich Focht	NEC	VO support update, Checkpointing Linux modifications
0.10	17/11/06	Christine Morin	INRIA	conclusion and executive summary
0.11	20/11/06	Oscar D. Sanchez	INRIA	Some minor changes
0.12	27/11/06	Massimo Coppola	CNR	shortened executive summary
0.13	29/11/06	Haiyan Yu	ICT	added Glossary
0.14	29/11/06	Massimo Coppola	CNR	shortened exec. summary, minor changes
0.15	09/12/06	Massimo Coppola	CNR	Changes to abstract, introduction, conclusions.
0.16	11/1/07	Yvon Jégou	INRIA	updated the definition of Virtual organization from D3.5.2

Abstract

According to the description of work [19, Annex 1], the XtreemOS operating system is composed of two parts: **1)** the foundation layer (XtreemOS-F), which is a modified Linux system, embedding Virtual Organization (VO) support mechanisms and providing an appropriate interface to implement **2)** the high level operating system services, (XtreemOS-G). This document presents the specification of Linux-XtreemOS (Linux-XOS), the instance of XtreemOS-F for grid nodes consisting of a single computer (PC). It results mainly from the joint work that has been carried out by INRIA, CCLRC, CNR, NEC, SAP, ICT and TID since the beginning of the project (June 2006) in Workpackage WP2.1, and from discussions with other XtreemOS partners, especially those involved in SP3 sub-project (XtreemOS-G specification) and in WP4.2 workpackage (application requirements).

The presentation follows two main directions: the specification of features to support Virtual Organizations, and that of features to support application checkpointing. In both cases, we put an emphasis on maximizing the acceptance likelihood of the XtreemOS design from the Linux community, by leveraging existing well-established standards and accepted tools in the XtreemOS-F software architecture.

The goal of VO support in XtreemOS is to provide mechanisms to set up and manage VOs in a scalable and flexible manner, and mechanisms which ensure access to various resources with fine-grained, mandatory access control without sacrificing site autonomy. We have identified several issues to be considered (§2.1), by analyzing requirements gathered from XtreemOS use cases, and also from other EU funded research projects (EGEE and Akogrimo) as well as the OGSA standard from GGF. The analysis of the state of the art (§2.2) on management of VOs makes evident that despite significant advances, there are still open issues with respect to **a)** scalability of in-the-large VO management, especially for short-lived and dynamically changing VOs, **b)** ease of management of VOs and VO identities, and **c)** security and VO policy enforcement at the node and site level, where current VO middleware cannot fully leverage the native OS.

We distinguish two levels in VO management: VO level (administration) and node level. VO-level management is performed by XtreemOS-G services (WPs 3.2, 3.3, 3.4 and 3.5), and it includes distributed information management for membership tracking and accounting of users and resources. The first of the two tasks of WP 2.1 is to add local mechanisms for recognizing, controlling and enforcing usage of global Grid entities to the standard Linux kernel, which is unaware of Grid entities. We have identified the main responsibilities of node-level management, which cover the areas of (Grid) identity management, resource access granting, VO policy checking, auditing, and enforcing. In this document (see §2.3) we give an overview of the basic implementation of VO support in XtreemOS, and how the

needed mechanisms can be developed, starting from practical definitions of *Grid identity* and *Grid session* that are suitable to a scalable implementation.

Our approach aims at minimizing changes to Linux code, especially within the kernel, and keeping them localized. PAM plugin-based authentication, static *and* dynamic identity mapping to local user/group ids, kernel-level key retention, and ACL mechanisms can be exploited to ensure that the VO model is flexible, secure, efficient as needed, and easily sustainable from the software engineering viewpoint. We are investigating on further synergies with existing mechanisms of security enhancement for Linux, such as the Linux Security Module (LSM) framework [54] and operating system level virtualization techniques [21, 9, 36], in order to refine access control and enforcement mechanisms in advanced versions of XtreemOS (to be implemented in the second half of the project, after M18).

We have analyzed the potential impact of the proposed specification on the Linux code (§2.4), identified its possible shortcomings, and compared it with other approaches (§2.5). Overall, by extending the Linux operating system with built-in VOs support, XtreemOS can provide outstanding performance and enhanced security, while minimizing administration costs of VOs compared with existing middleware solutions for VO.

Chapter 3 of this deliverable discusses how Linux-XOS will implement methods and interfaces to checkpoint and restart applications. Taking into account the requirements related to checkpointing functionalities, originated from all research units of XtremOS (see §3.1), WP 2.1 has performed an evaluation of existing approaches to local (non Grid-aware) checkpointing (§3.2). The state of the art includes quite different approaches with respect to the implementation and the boundaries of the checkpointed units, and to the degree of collaboration required from the applications (e.g. the presence and complexity of an application oriented checkpoint API).

The XtreemOS approach will involve a hierarchical decomposition of the implementation (§3.3) into a kernel checkpointer, a system-level checkpointer and a grid-wise checkpointer. The two former checkpointers are implemented in XtreemOS-F, while the latter is a service in XtreemOS-G, developed in WP3.3 as a part of the application management service.

From this analysis, a design of the local checkpointing functionalities has been derived (§3.4). Kernel checkpointing of processes in Linux-XOS will be based on BLCR, which is one of the most advanced open-source implementations of a checkpoint system for Linux. Additional features are needed to comply with XtreemOS requirements, namely **a)** explicit management of shared libraries used by each application unit, **b)** proper management of security context w.r.t. VOs within each snapshot unit, **c)** providing information at unit restart about environmental changes.

The mechanisms that allow a process to be notified of checkpoint events by the kernel are being considered for addition to the kernel interface. More investigation is needed to properly design the call-backs which will coordinate units of a Grid application at restart time, in order to restore network connections and other forms

of shared resources. BLCR was chosen as a starting point, according to the global strategy of minimal kernel changes, as the porting effort is confined to a kernel module and may exploit collaboration with Berkeley Labs. Major improvements of checkpointing functionalities in XtreemOS will come from exploiting XtreemOS (Grid) filesystem features to enhance BLCR behavior.

The XtreemOS system checkpointer is viewed as an implementation of the `job_service` described in the Simple API for Grid Applications (SAGA [29]), which is parametric with respect to choices like checkpoint frequency and storage policy. By M18, we plan to be able to checkpoint/restart applications consisting of a single application unit (that is to say, applications running entirely on a single Grid node). Checkpoint/restart of applications made up of multiple application units is considered as an advanced feature to be further investigated after M18.

Contents

Chapter 1

Introduction

The XtreemOS Grid Operating system will offer a native support for Virtual Organizations (VOs) in Linux. As described in the description of work (Annex 1) [19], the XtreemOS operating system is internally composed of two parts: XtreemOS foundation, called XtreemOS-F, and XtreemOS high level operating system services, called XtreemOS-G. XtreemOS-F is a modified Linux system, embedding VO support mechanisms and providing an appropriate interface to implement XtreemOS-G services.

XtreemOS-G is implemented on top of XtreemOS-F as a transparent layer for the user. More accurately, this is the point of view of VOs, and users within a Grid are to be supported exploiting high-level mechanisms which rely on those provided by the single nodes. XtreemOS-G comprises services for security, data and application management, all of them based on a common infrastructure designed to provide highly available and scalable services.

There are three different flavors of XtreemOS-F depending on the kind of grid node considered: single computer (PC), cluster or mobile device. This document presents the specification of Linux-XOS, the instance of XtreemOS-F for grid nodes consisting of a single computer. It results mainly from the joint work that has been carried out by INRIA, CCLRC, CNR, NEC, SAP, ICT and TID since the beginning of the project (June 2006) in Workpackage WP2.1 and from discussions with other XtreemOS partners, especially those involved in SP3 subproject (XtreemOS-G specification) and in WP4.2 workpackage (application requirements).

The Linux-XOS core consists in the existing Linux operating system. Linux will be modified (with kernel patches), extended (with kernel modules) and configured (enabling optional components and/or exploiting the framework provided by the Linux system) according to the need to support virtual organizations, and in order to provide the features and API needed to implement the XtreemOS-G services.

In the following we describe our approach to XtreemOS-F specification and design, then the actual work of WP 2.1 on the specification of Linux-XOS will

be introduced according to its two main directions: the specification of features to support VOs and the specification of features to support application checkpointing.

1.1 XtreemOS-F Design Strategy

As we have mentioned, our analysis involves evaluating changes and modifications to standard Linux systems which occur at quite different levels of the system implementation. Since the XtreemOS project in the overall aims at introducing new, Grid-enabling features into Linux, then, beyond the technical aspects of soundness, performance and scalability of design, all choices related to the specification of XtreemOS-F need to be evaluated in terms of impact on the Linux code base and on the underlying open-source software development process.

In order to enhance the impact factor of the XtreemOS project on Linux and to ease acceptance of the new features into the standard code base, the approach we follow aims at being minimal with respect to kernel code patches, and at keeping required changes localized in dynamically loadable kernel modules. We exploit as much as possible existing community standards and kernel APIs. This reduces the pressure to get VO related changes accepted by the kernel developer community.

The rationale is that widely accepted and useful features are maintained by the whole Linux community, less commonly used or esoteric features are maintained by smaller and smaller sub-communities, eventually by their originators. Whenever change maintenance needs to be carried on in synch with the evolution of system code, the effort required from those communities becomes a practical limit. By adopting existing features, pushing small changes into the mainstream and ensuring that the maintenance of more complex contributions is relatively independent form kernel evolution, we enhance the likelihood that XtreemOS-F, in its maturity, will successfully exploit the collaboration of the open-source software development community.

1.2 Virtual Organizations in XtreemOS

A Virtual Organization is a temporary or permanent coalition of geographically dispersed organizational units (from individuals to entire organizations) that pool resources, capabilities and information to achieve common objectives.

The XtreemOS approach is to propose operating system extensions to augment user and resource management mechanisms in existing operating systems, in order to natively support cross-domain resource sharing. New mechanisms are needed to support scalability to a potentially large number of users and resources within a grid.

The cost of administering and operating a VO (e.g., adding or removing nodes, changing access policy, authenticating and authorizing users) should be minimized and possibly bound to a known value rather than to grow with the number of users

and resources participating in the VO. Moreover, the dynamicity of users and resource usage needs to be handled in a flexible way.

The XtreemOS aim of providing the abstraction of a conventional operating system on top of a collection on nodes produces another complex set of requirements. We need strict enforcing of VO policies on Grid resources, and to develop account mechanism for global grid identities which are compatible with the local Linux ones. We have to harmonize and let interoperate the DAC (Discretionary Access Control) and MAC (Mandatory Access Control) models, primarily taking into account that VO administration is a source of potential security holes.

A VO security policy should be harmonized version of the desired security features of the VO and the security policies of its partner institutions. Security gaps should be checked for at VO creation time, avoiding any chance of malicious activities. The resilience of VOs should be ensured in the face of a set "legal" modifications in the hardware (hardware upgrade) and the software (installing new applications) of Grid nodes, without having to reset the whole VO each time. More important, the security and resource management mechanisms of VOs should be able to stand a certain degree of temporary and permanent resource failures. Finally, VO support should include monitoring mechanisms.

XtreemOS will enhance the Linux operating system with mechanisms for decentralized management of users and resources, based on global user identifiers together with domain-independent access control models. It is needed to specify where and how to incorporate VO support features into current OS kernels. The overall principle is to reduce management and maintenance costs for VOs when crossing domains and to provide strengthened security support for applications, without compromising the execution efficiency.

When complex grid applications are deployed across multiple grid nodes, access control to local kernel objects has to be decoupled from that of Grid resources. This approach ensures modularity at the XtreemOS-G level, as Grid resources are managed only according to Grid credentials, and saves deep modifications to the Linux kernel in XtreemOS-F. A powerful but practical definition of VOs is required, and efficient Node-level management functionalities are needed to ensure that XtreemOS-F nodes will suit as VO building blocks.

1.3 Checkpointing in XtreemOS

On top of XtreemOS, a Grid application is executed on one or several Grid nodes. Such an application is composed of application units, any application unit being executed on a single Grid node. Checkpointing of applications is generally required due to the dynamic nature of Grid computing platforms, and in the case of XtreemOS is also due to the dynamic nature of virtual organizations. Application units running on a grid node may need to be moved to another node during their execution, or to be restarted, as a consequence of node failures, varying resource load, and changes in the policies of the involved VOs.

We should be able to restart an application unit running in the context of a VO on a different node with the same architecture within the given VO, eventually in a different administration domain. It is thus important to develop methods and interfaces to checkpoint/restart applications. XtreemOS approach is to extend Linux to integrate kernel level process checkpointing and to enable application unit checkpoint/restart.

Application checkpointing in XtreemOS is hierarchically decomposed into three levels of checkpointers: a kernel checkpointer, a system-level checkpointer and a grid-wise checkpointer. The two former checkpointers are implemented in XtreemOS-F, while the latter is a service in XtreemOS-G, developed in WP3.3 as a part of the application management service.

The kernel checkpointer adds two functionalities to a standard Linux kernel,

- it brings a process to a checkpointable state,
- it saves a snapshot of the state of a process to a file descriptor.

XtreemOS-F will need to augment the kernel and single system APIs to support Grid-enabled checkpointing. New APIs will have to be carefully designed according to the overall approach of minimal and local code changes, and existing systems and tools for checkpointing at the different levels have to be analyzed. This is especially important for the mechanisms providing restarted processes with information about changes in the environment (process id, IP address, hostname) and for those which allow restarted application units to coordinate and bring Grid-shared resources (e.g. network connections) to a correct state.

1.4 Document Structure

The outline of the document is as follows. Chapter 2 is devoted to the presentation of Linux-XOS specification regarding features to support virtual organizations. In chapter 3, the process checkpointing and recovery mechanisms to be provided by Linux-XOS are presented. In these two chapters we follow a common approach. We first give a summary of application requirements with respect to the specific feature, we present the related state of the art, then we describe the XtreemOS approach, and finally we discuss the foreseen modifications to the Linux operating system.

Section 2.1 thus lists VO related requirements from XtreemOS use cases, as well as requirements gathered from other EU funded research projects (EGEE and Akogrimo) and the OGSA standard from GGF. Section 2.2 discusses the state of the art of VO management. The analysis leads to the XtreemOS definition of VO support in section 2.3, whose impact on Linux code is discussed in section 2.4, and which is compared with other approaches in section 2.5.

Section 3.1 lists requirements related to checkpointing, which are evaluated against the state of the art in section 3.2. The reference XtreemOS architecture

for checkpointing is described in section 3.3, and the design of the XtreemOS-F checkpointing functionalities is given in section 3.4.

Chapter 4 concludes the deliverable, summing up the results and stressing on the main features of Linux-XOS, on the interactions between XtreemOS-F and the higher level services, and stating what we plan to achieve in the first basic version of XtreemOS to be delivered by the end of 2007.

Chapter 2

Virtual Organization Support in Linux

2.1 Application Requirements

Linux-XOS specification is mainly based on the applications requirements stated by the different partners. First, a template document was elaborated for partners providing applications to manifest their particular requirements. This template was filled out by the different groups according to their applications' requirements. After that, the requirements were consolidated into a single document that will be used for Linux-XOS specification.

The applications used to elaborate the requirements consolidation are described in Table 2.1. These applications include a wide variety of fields (from finance modelling to personal messaging) and behaviours (from interactive to CPU-demanding) that will provide a good foundation to cover all the necessary requirements for Linux-XOS specification.

This requirements consolidation document is basically composed of two groups of requirements:

- The first group of requirements (from R1 to R17) describes general requirements.
- The second group of requirements (R19 and above) are those requirements associated to a particular workpackage.

2.1.1 General Requirements

General requirements are those requirements which could not be directly associated to any workpackage because they refer to general aspects of XtreemOS. In this section, only general requirements associated to VO and resource management are explained. Requirements related to checkpoint of applications will be addressed in section 3.1.

Application	Group	Description
SPECweb2005	BSC	SPEC benchmark for evaluating the performance of www servers
GSfastDNAml	BSC	Best-tree search in a set of sequences of taxa
Elfipole	EADS	Software chain performing acoustic wave propagation simulation
HLA	EADS	Real-time simulation communicating through HLA middleware
MODERATO	EDF	Simulation of radiographic inspection and complex inspection configuration
SIMEON	EDF	Simulation of energy generation planning
ZEPHYR	EDF	Solving of a 2 dimensional nonlinear unsteady viscous Bürgers equation
SRC	EDF	Secure Remote Computing
WebAS	SAP	Application platform for development and execution of web services and applications based on J2EE and ABAP
DBE SF	T6	Service discovery and execution support for SMEs (Service Factory)
DBE ExE	T6	Service discovery and execution support for SMEs (Service Execution Environment)
Tifon	TID	Messaging application
JobMa	TID	Job Management application
Wissenheim	ULM	Virtual presence application
Galeb	XLAB	Finance modelling framework for nonlinear time series analysis and prediction

Table 2.1: Applications used for the requirements consolidation.

- **R16: XtreemOS VOs shall manage a large number of users**

  > "The number of users within a VO is very different for the applications considered. About half of the applications can be executed with at most 4 different users whereas the other half demands to manage several thousands of users, which may also concurrently work with the application."

  The specification of users management mechanism will take into account this requirement by assuring that VOs will support a large enough number of users. Rather than specifying a minimum number of supported users, the specification will leave this decision to the particular implementation.

- **R18: XtreemOS VOs must provide role management**

  > "XtreemOS must be able to manage users through roles, each role having its own rights. Users must be able to read, write, change and delete files and to execute applications. Administrators need to give permission for installation, maintenance and to manage accounts (add/remove accounts, change of permissions)." [...]

  The complexity of large environments such as a computational Grid requires the use of sophisticated methods for controlling access to resources. The use of roles in XtreemOS will make it possible for system administrators to control access to a given resource by assigning roles to the different users of the system. Permissions to access a given resource will then be assigned to the different roles. In this way, establishing an access control policy will be a matter of assigning roles to users.

2.1.2 WP-specific Requirements

These are requirements which are clearly related to a particular XtreemOS workpackage. Hence each one of these requirements will be handled by the appropriate workpackage.

- **R19: XtreemOS has to provide the means to manage VOs**

  > "A Virtual Organization must be manageable. This means an authorized user can create, change and destroy a Virtual Organization. Three different roles will be involved in managing VOs: domain administrators, global VO administrators and global users." [...]

Domain administrators maintain a pool of computing resources that are integrated into a VO. *Global VO administrators* are allowed to compose VOs from the resources provided by various domains. They are also responsible for creating global user accounts and for configuring and modifying the permissions that global users have within a VO. *Global users* also require the right to manage a VO. The respective (restricted) permissions are granted by the VO administrator. This allows for instance that a VO is created when an application is started and the VO is destroyed after the application end.

- **R20: VO user accounts have to be independent from local user accounts**

  > "A person can have a user account in her/his local domain A and a global user account in a VO comprising domains B, C and D. To obtain a global user account it is not necessary to have a local user account in one of the domains belonging to the VO."

  According to this requirement, local and global user accounts will be independent. Users will be able to log into their local account, but they will only be able to access the Grid when logged into a global account. Local accounts will be maintained for backwards compatibility with traditional UNIX systems.

- **R21: XtreemOS allows to establish and manage hierarchies of VOs**

  > "Hierarchies of VOs can either be created from scratch or by dynamically establishing sub-VOs. [...] it must be possible to establish VO2 as a sub-VO of VO1, with the same or limited rights."

  Virtual Organizations in XtreemOS will have a hierarchical structure. In this manner, the creation of sub-VOs (VO2) inside higher-level VOs (VO1) will be controlled by the higher-level VO (VO1) administrators. VO administrators will be able to specify who can create sub-VOs inside the VOs that they administer.

- **R22: XtreemOS allows to dynamically change the composition of VOs during application runtime**

  > "It must be possible to change the composition of resources within VOs while applications are executed. This dynamic adaptability is needed e.g. if certain computing or communication resources fail. In this case the unavailable resources need to be automatically substituted by alternative resources (also from other domains)."

  As needed by the applications, it will be possible to modify the structure of a VO without needing to stop running applications. XtreemOS will provide

mechanisms to help applications detect when the Grid structure has changed (i.e. resources have been added or removed).

- **R23: The lifetime of a VO must be guaranteed**

  > "The lifetime of a VO (as required by the applications) may range from a couple of days up to infinite (i.e. not known in advance). Therefore it must be possible to:
  > a) Guarantee the lifetime of a VO for a specified amount of time.
  > b) Guarantee the lifetime of a VO until a notification that the VO is not required anymore."

  This requirement involves mechanisms for allowing an application to specify the amount of time during which the application will need the VO services. Also, XtreemOS VOs will have to guarantee their lifetime until a notification is received.

- **R24: XtreemOS must allow to establish multiple VOs on the same node within specified constraints**

  > "VOs comprise nodes from different domains. An overlapping of VOs is allowed. This means that it must be possible that a node is contained in multiple VOs. The domain administrator, however, is allowed to restrict the maximum number of VOs to which a certain node can belong to." [...]

  A single Grid node is allowed to be considered a computing resource for one or several Virtual Organizations. When several Virtual Organizations can access the same computing node, isolation mechanisms must be specified that avoid conflicts between these Virtual Organizations.

- **R25: A VO management interface has to be provided**

  > "The management of VOs must be possible by an API (Application Programming Interface), a TUI (Textual User Interface) and a GUI (Graphical User Interface). The management interface must also provide means for monitoring the VO, e.g. information on the effectiveness of the changes made on VOs."

  VO management is a critical aspect of XtreemOS. In any grid environment, it is very important that all the correct permissions are set, policies don't have any security hole, and that administrators have everything under control. Hence, XtreemOS needs to provide administrators with all the possible means to manage VOs. The API will be designed so that applications can

take advantage of all of the functionality of XtreemOS VO management. The TUI will provide high-level VO-administration mechanisms accessible from text-based terminals. Finally, the GUI will provide an easy-to-use graphic interface that will allow system administrators to add users or resources, change permissions, roles and/or policies.

- **R26: VO management actions must be completed within a specified maximum amount of time**

  > "Various applications require that a VO can be created, changed and destroyed within a certain maximum amount of time. In some cases (HLA, SIMEON, Wissenheim) this response time needs to be a couple of seconds or at most 10 minutes. It is therefore necessary that global users can in advance define how fast management actions need to be performed with respect to each application (to be agreed with the VO administrator)."

  A VO management action can be for instance creating a new VO, adding a new user to a VO, changing a certain resource's access policy, managing groups or roles, etc. Applications need to know how much time a VO management action will take, as they will be permitted to perform these actions. It is important that this time be as little as possible, and mechanisms will need to be provided so that a maximum amount of time for each action can be specified.

- **R27: XtreemOS has to support communication between VOs**

  > "The applications require exchange of information between VOs by means of messages (also instant messages) and shared memory."

  Even though a certain degree of isolation will be enforced between Virtual Organizations, some applications require that communication between Virtual Organizations be possible. Thus, mechanisms will be provided to allow this communication.

2.1.3 Existing Middleware Requirements

Besides the requirements from the applications, we have also analyzed current middleware projects in order to extract requirements that XtreemOS should take notice of. The main reason for performing this search in other projects is that this guarantees that requirements that may be important for Linux-XOS, but have not been identified by applications (see Table 2.1), are not left out of the specification. From these projects, only those requirements that are related to VO and resource management will be analyzed in this section.

Requirements from EGEE Project

Enabling Grids for E-sciencE [2] is an EU-funded project aiming to provide researchers in academia and industry with access to computing resources regardless of their geographic location.

Applications in the EGEE project have generated a large requirements database [3]. These requirements are classified into the following sections:

- Functionality
- Interoperability
- Performance/Metric
- Reliability
- Security
- Use Case

From the requirements in this database, we have taken the ones that VO and resource management in Linux-XOS should support, but have not been identified by our applications. These requirements will also be considered in Linux-XOS specification. We represent these requirements, literally extracted from EGEE database:

- **#100449 VO-wide resource allocation to users (Use Case)**
  Set/Modify Resource Allocation for groups/users of a VO
- **#100470 User identification policy for grid portals (Security)**
  The user identification policy behind a given portal certificate must be easily obtained and completely documented. (E.g. truly anonymous access, access with a known email address behind, access with username/password, etc.)
- **#100537 Information system bootstrapping (Functionality)**
  There must be a simple deterministic procedure to find the entry points to the information system (bootstrapping problem).
- **#100538 Information on services accessible to the users (Functionality)**
  It must be possible to obtain information about all grid services the user is authorized to use (in particular, VOs, RBs, CEs, SEs...) through a well known and unique procedure. The information system bootstrapping issue (see requirement #100537) should be solved for this.
- **#100550 New VOs creation (Functionality)**
  Creation of new VOs must be a lightweight process enabling integration of new user communities. It must not take more than one day to set up a new VO (including VO server set up and configuration, a mechanism for users to register to the new VO, and VO-specific services start-up).
- **#100551 VO-specific services creation (Functionality)**
  The setting-up of VO-specific services (VO LDAP, RMC, etc) must be integrated in the VO creation procedure so that when a new VO is created and users can register to it, they will find all basic services needed for basic grid usage.

- **#100552 VO authorization (Functionality)**
  The policy to attribute resources to new VOs must be well established and a minimal amount of resources on which the VO is authorized must be allocated when a VO is created so that new user find resources for proper grid usage.
- **#100553 User authorization (Security)**
  A user must always be granted access to grid services he/she is authorized to use.
- **#100554 User denial (Security)**
  It must be possible to deny access to some grid services to a specific user. The most important part of this requirement, is to be able to deny access to all services at once (e.g. by preventing the user from obtaining a valid credential). A finer, service per service, denial capability would be an improvement.
- **#100558 Multiple VOs affiliation (Functionality)**
  A user must be able to register with multiple VOs and use the rights from those VOs simultaneously.
- **#100559 Roles and groups within VOs (Functionality)**
  It must be possible to define roles and groups within a VO for fine-grained authorization control. Group of users are used for file access control (all users within a group will have the same rights as specified for groups in the ACLs). Roles define the user capabilities (e.g. maximum priority level, administrative right on files, etc).
- **#100572 Select wished VO rules (Security)**
  Tools shall be available to allow the user to select and de-select which VOs and roles they wish to enable for the current session. This shall include the ability for each user to select a default VO and role which should automatically be set up at the beginning of each session.
- **#100643 Initiating revocation method (Security)**
  The methods of initiating revocation shall be published by authorities issuing revocatable credentials.
- **#100644 Initiating revocation (Security)**
  It shall be possible for authority parties to initiate revocation.
- **#100662 Member of any num of VOs (Security)**
  Users shall be members of any number of VOs.
- **#100682 Remove user from VO (Security)**
  It shall be possible to remove a user from a VO.
- **#100683 Remove user roles (Security)**
  It shall be possible to remove one or more roles from a user.
- **#100719 VO membership confidentiality (Security)**
  A user's membership in VOs shall be confidential information.

- **#100720 VOs decide membership (Security)**
  The VO shall be able to decide user membership policy and user authorization policy.

Requirements from Akogrimo

Akogrimo [1] is a EU-funded research project that aims to advance the pervasiveness and mobility of Grid computing across Europe. Akogrimo proposes an accounting session model, introducing the concept of "Session ID". Given that this session model could be useful for XtreemOS, we have included here certain requirements related to the "Session ID":

- **Session ID must be globally unique**
  To uniquely identify a session, a session ID must be globally unique which means unique in time and location. The might be relaxed to unique within a given time period or within a certain spatial scope. For instance, if we have to audit session data for 10 years the session ID must be unique within this period. If it is assured that a session does not cross certain boundaries the uniqueness requirement may be relaxed to "unique within these boundaries".
- **It shall support fine granular auditing**
  The session ID must allow for the finest grained auditing possible of a provided service.
- **It must support flexible accounting**
  The binding model of the session IDs should not limit the accounting and auditing capabilities.
- **The model should be efficient and scalable**
  The generation and linkage of session IDs must be sufficiently fast compared to the overall process of service authentication and authorization to avoid unnecessary latency.

Requirements from OGSA Specification

The specification of the Open Grid Services Architecture (OGSA) [25] is a document produced by the OGSA working group within the Open Grid Forum (formerly known as the Global Grid Forum). This document defines a series of requirements to support Grid systems focusing on application in e-science and e-business.

These requirements are classified into the following sections:

- Interoperability and Support for Dynamic and Heterogeneous Environments
- Resource Sharing Across Organizations
- Optimization
- Quality of Service (QoS) Assurance
- Job Execution
- Data Services

- Security
- Administrative Cost Reduction
- Scalability
- Availability
- Ease of Use and Extensibility

Some requirements that may be of importance for VO and resource management in Linux-XOS are those related to interoperability and resource sharing across organizations. We summarize these requirements in the following paragraphs:

- **Resource discovery and query**
  Mechanisms are required for discovering resources with desired attributes and for retrieving their properties. Discovery and query should handle a highly dynamic and heterogeneous system.
- **Standard protocols and schemas**
  Important for interoperability. In addition, standard protocols are also particularly important as their use can simplify the transition to using Grids.
- **Global name space**
  To ease data and resource access. OGSA entities should be able to access other OGSA entities transparently, subject to security constraints, without regard to location or replication.
- **Site autonomy**
  Mechanisms are required for accessing resources across sites while respecting local control and policy.
- **Resource usage data**
  Mechanisms and standard schemas for collecting and exchanging resource usage (i.e., consumption) data across organizations for the purpose of accounting, billing, etc.

2.2 State of the Art

This section presents a survey on state-of-art approaches for distributed resource sharing by means of Virtual Organizations (VOs) [24] that may cross multiple administrative domains. A VO, known as a virtualized coupling of resource consumers and providers from real organizations, is a widely accepted model to build a boundary for coordinated resource usage and access control under specific sharing rules, which are agreed by multi-institutional parties (domain administrators). A more detailed state-of-art report on current VO frameworks and security services is presented in D3.5.1 [7]. This survey examines the most related works concerned with WP2.1, including how user identities are managed in a cross-domain environment like a VO, how various authorization frameworks are employed by current Grid middleware to build VOs, and how resource access control on nodes of a VO

is enforced and isolated based on a variety of extensions to local operating systems. Related standards and open issues are also discussed.

2.2.1 Cross-Domain User Identity Management

User identity is a set of qualities or characteristics that distinguishes a user from other users within the same group (e.g. a node). In a distributed network environment where nodes may come from multiple administrative domains, management of user identities needs to be well coordinated to simplify system administration, to facilitate resource sharing and to ensure network security. The basic principle followed by existing approaches is that each user needs only one identity for accessing all nodes, or more precisely, to support the Single Sign On (SSO) feature.

Centralized Identity Management

Though scalability issues such as single point of failure and overload coexist with centralized approaches for managing user identities, they are extensively supported and used in current operating systems. NIS/YP [47] is a RPC-based service that maintains a central database of configuration data for a group of network machines in a NIS domain (similar to a Windows NT domain). User identity information (e.g. passwd,group,alias) could be stored/retrieved by NIS clients from the NIS domain server for authentication. LDAP [18] provides a better repository service to store information as a hierarchical, directory-based structure compliant to X.500 specification. Authentication systems could rely on LDAP to centralize user information in a standard way, which often reflects geographic or organizational location of users. LDAP could run on top of TLS for enhanced security. Kerberos [5] has evolved into a standard based strong authentication system based on using symmetrical encrypting algorithms such as DES and 3DES. The basic idea of Kerberos is to have a central Key Distribution Center (KDC) that is responsible for allocating temporary tickets among users and application servers who need secure communication.

Distributed Authentication

A unique feature of distributed authentication approaches is that validation of user identities could be done offline by asymmetric cryptographic techniques, without the need to interact with online network authentication services. Public Key Infrastructure (PKI) [31] is a set of IETF standards (X. 509 series) that define how the certificates and CAs must work together to constitute a distributed authentication framework. The Grid Security Infrastructure (GSI) [15] software is a set of libraries and tools based on extensions to PKI, i.e. the newly designed proxy and delegation mechanism, to enable users and applications to access remote resources securely. A proxy credential has its own private key and certificate, and is signed using the user's long-live credential. Delegation is the creation of a new set of

proxy credentials on remote sites from existing set of proxy credentials on local sites. The creation is accomplished via a GSI-authenticated conversation. By such way, users' identities could be relayed across multiple sites without the intervention of users. Delegation can be chained to achieve the Single Sign On property.

Federated Identity Management

The most significant feature of federated identity approaches is keeping users' privacy at a maximum level while supporting scalable distributed authentication and authorization. The basic idea explored by Shibboleth [43] is the federated administration by the alliance of trusted home sites for users. Users are registered only at their home site rather than at each resource provider. Resource providers rely on users' home sites to provide identity information as well as attributes about users. In Shibboleth an opaque handle is returned as a response for an authentication request, which may not carry any user privacy information such as his identity. A user is allowed to select a subset of his attributes to present to providers. As a whole Shibboleth is a set of specifications to ensure the secure transfer of user attributes among home sites and resource providers. The Liberty Alliance [6] is another similar work to Shibboleth, which also defines suites of specifications for federated identity management and web services communication protocols.

2.2.2 Authorization Frameworks for VOs

VO management services in current Grid systems put focus on authorization frameworks that maintain membership of users, nodes, policies and use conditions of resources in a VO. The common objective of these frameworks is to provide flexible, scalable, and fine-grained authorization support for VOs. The diversity of their implementations is embodied in different message sequences (e.g. agent, push and pull described in RFC 2904 [32]), different places to convey attributes (e.g. critical or non-critical section of a proxy certificate), different policy models (e.g. MAC or RBAC), etc. The following paragraphs are not for an orthogonal classification but only reflect a collection of similar works respectively.

VO/Community Authorization Services

Virtual Organization Membership Service (VOMS) [14] is a centralized authorization framework originally developed in European DataGrid and DataTag project. It is designed to support VO-level fine-grained authorization based on dividing VO users into groups or subgroups and assigning them different roles and capabilities. Basically VOMS adopts a push based message sequence. Before accessing a resource, a user and the VOMS server authenticate each other and then the user sends requests to one or more VOMS servers to get his attributes information. VOMS server distributes the Attribute Certificates (AC) [23] it creates to the users themselves, who are then responsible for presenting their certificates to resources.

Community Authorization Service (CAS) [35] exploited the idea that resources fully delegate their access rights to the community service rather than individual users. The community service acts as an agent for users by distribute a particular set of rights to providers according to users' identities and the community policies. CAS extends GSI proxies to *restricted proxies* in which delegated rights information is encoded in a critical X.509 extension. PRIMA [38] is another VO management framework similar to VOMS in many aspects. The light-weight design of PRIMA enables the creation of small and dynamic communities via a set of tools and extension modules without deploying community servers. The PRIMA system also uses X.509 Attribute Certificates (AC) to securely bind privilege attributes to subjects and policies to resources.

Policy-based Authorization Infrastructures

Akenti [50] is an authorization infrastructure developed at Lawrence Berkeley National Laboratory. It is designed to address complex authorization problems involving multiple administrative domains and multiple stakeholders (a stakeholder is an X.509 "source of authority"). For authentication, Akenti relies on X.509 certificates and the SSL/TLS protocol to securely authenticate a user, like most PKI-based systems. For authorization, Akenti uses a pure pull model. When a resource request comes, the Akenti policy engine collects all relevant certificates from both the user and the resource, and derives the user rights from them. It is the server side that contact all authorities once the user gets authenticated. One potential problem within Akenti is its policies are expressed using a proprietary XML format rather than X.509 based. With many common features with Akenti, PERMIS [22] is another policy-based authorization system, from the EC funded PrivilEge and Role Management Infrastructure Standards validation (PERMIS) project. Unlike Akenti's distributed and hierarchical policies, the policy in PERMIS is a single attribute certificate stored in a LDAP directory. It supports the role based access control (RBAC) paradigm, which means, PERMIS infers the access right (roles and attributes) according to the given user's DN, a resource and an action. It supports classical hierarchical RBAC in which superior roles inherit the privileges of subordinate roles in the hierarchy.

Open Authorization Frameworks

New version of Globus Toolkit, GT4 [4], is geared towards a service-oriented grid middleware following the WSRF and OGSA specification series. It provides a pluggable authorization framework with callouts which allows third party SAML-compliant authorization services, including Policy Information Points (PIPs) and Policy Decision Points (PDPs), to be chained into GT4 containers. These callouts could be used to integrate existing mature authorization frameworks to enhance security of Globus Toolkit, which is exemplified by GridShib [48] project. GridShib is a joint effort aims to provide attribute based authorization in the Globus Toolkit

by inter-operating with Shibboleth, the well designed and deployed security architecture for Internet2. In GridShib, Globus Grid Service Providers (GSPs) are capable of securely request a user's attributes from the Shibboleth Identify Provider (IdP), with the help of a GridShib PIP plugin. Users are identified by distinguished names (DNs) in GT4, while in Shibboleth by opaque handles. GridShib addresses this mismatching problem by developing a DN mapping plug-in for the Shibboleth IdP.

2.2.3 Resource Management and Access Control in VO nodes

The policies specified by a VO, such as security, resource limitations, scheduling priorities and whatever attributes imposing regulations on how shared resources could be used by members of a VO, will be finally checked and enforced at resource nodes (sites). The main challenge here is the isolation among different users' accesses to the same node and also multiplexing usage of the same node from different VOs. At the same time, the autonomy nature of a node must allow domain administrators to have the final control of resources which precedes any VO policies. This problem is also known as workspace management [11] . Generally there are two approaches for protection and isolation of VO accesses in local operating systems: account mapping and virtual machines.

Account Mapping

Account mapping is proved to be a light-weight but efficient approach for isolation in which grid user credentials (e.g. certificates, attributes, etc.) are mapped onto local credentials (e.g. user accounts, ACLs, etc.). Access control enforcement fully relies on traditional account-based separation within local operating systems. The default security configuration file in GSI, *grid-mapfile*, specifies a list of grid users who are allowed to access local node and which local account will be mapped to each grid user. Once a grid user's DN is in the list, he/she has all the rights of his/her mapped local account in the node. This approach provides limited functionality and it lacks the necessary scalability and flexibility for supporting large-scale VOs. Local Credential Mapping Service (LCMAPS) from VOMS [14] is an enhanced framework that can load and run one or more "credential mapping" plugins. For example, it supports the dynamic creation of grid-mapfiles via EDG *mkgridmap* utility. And it also supports plugins of pool accounts, which enable the local mapped accounts to be allocated on-the-fly from a pool of generic leased accounts. The leases are revoked after a specified time and are available to other incoming grid users. Pool accounts could simplify the administration of a node serving large number of grid users, however, the revocation of accounts may cause inconsistency issues for users' generated files. Based on account mapping, attributes specified in a VO policy could be translated into local resource capabilities such as permission bits and POSIX 1003.1e ACLs of files.

Virtual Machines

An envision of virtual machine (VM) based approach for grid computing was justified in [44, 13], followed by numerous VM-based grid research prototypes including the virtual workspace [11, 34] of Globus Toolkit. VMs provide outstanding isolation properties by instantiating independent guest environments on a host. Besides this, VMs bring additional benefits such as full state checkpointing and migration, which facilitates the dynamic deployment of execution environments on demand. Current VM-based grid computing approaches are almost based on utilizing open source or commercial VM software techniques. These techniques adopt different levels of virtualization including emulation, paravirtualization and operating system-level extensions. VMware [12] is a commercial software suite allowing users to run multiple x86 virtual computers simultaneously on top of software emulators within one hosting operating system. User-mode Linux (UML) [21] allows multiple virtual Linux systems (guests) to run as an application within a normal Linux system (the host). It offers excellent security and safety without affecting the host environment's configuration or stability. Though paravirtualization techniques require the modification of guest OSes, they exhibit a fairly acceptable performance overhead as compared to standalone servers, which has been demonstrated by Xen [42]. Operating system-level support for virtualization, such as OpenVZ [9] and Solaris Container [36], introduces low overhead via a single-kernel approach, which allows running hundreds of virtual servers on a single physical server. Though VM techniques are mature enough to be used by the web hosting industry, the setup, deployment and management costs of VM environments are still high which hinders their prevalent adoption in grid computing.

2.2.4 Related Standards

SAML [40] aims to define a standard way to represent authentication, attribute, and authorization information that may be used in a distributed environment. OGSA Authorization WG [8] is an OGF working group to define specifications for basic interoperability and plug-ability of authorization components in the OGSA framework. Within its released documents, GFD.67 [53] identifies authorization requirements from a number of use cases and enumerates different models (push/pull) with which existing security frameworks (e.g. Akenti, Permis) can be used. GFD.66 [52] specifies standard formats of SAML extensions and prototypes of authorization services. A number of methods for requesting and encoding attributes already exist, including X.509 Attribute Certificates [23], SAML Attribute Assertions [40] and XACML Attributes [39]. GFD.57 [49] provides a suggestion to the Grid community on specifications for attribute assertions.

2.2.5 Summary and Open Issues

It is notable that there is no *silver bullet* solution for user management and access control of resources within a Virtual Organization. There is a consensus that

the Grid community start to put efforts on integrating complementary frameworks together on a basis of open standard interfaces (e.g. SAML). For example, the GridShibPERMIS project [51] is such an effort that brings together Globus Toolkit platform, Shibboleth's federated identity administration and Permis's RBAC policy engine to achieve a scalable and flexible management framework for VOs.

Though significant progress has been achieved with the development of flexible and scalable VO management frameworks and security services, there are several open issues left:

- The establishment of a VO is still a heavy-weight and time-consuming process. From the application perspective, dynamically composing resources into a VO or changing the VO composition on-the-fly could bring non-trivial administrative burdens to domain administrators, VO administrators and even grid users. From the user perspective, taking a VO as a virtual computer to use is far more difficult than using a traditional system when dealing with the complexity of identities, certificates and policies.

- There is still no perfect solution for node-level or site-level isolation of multiplexing accesses from different VO users. Account mapping is a widely accepted solution until now while virtual machine based approaches need more experiments to justify its overhead. Without the native OS support, account mapping approaches still face potential security issues when VO accesses cannot be differentiated, as well as scalability issues when VOs are dynamically changed.

2.3 Virtual Organizations in XtreemOS

2.3.1 Definition

A clear and unambiguous definition of important terms related to VO in XtreemOS is given as follows:

User A user is a person, corporation or equivalent entity who makes use of a computer system. A user in XtreemOS context is the one who accesses resources and services provided by a single node or a set of nodes which may be affiliated with a Virtual Organization.

To be precise, we differentiate the term *grid user* and the term *local user*. Only grid users are capable of accessing resources of VOs. Local users are normal Linux users not aware of VOs. For a person [1] who wants to use the Grid (i.e. resources in a VO), he/she must be a grid user, i.e., he/she must have a grid user identity (see next definition).

[1] It is possible for this person to have several grid user identities and several local user identities (Linux accounts) to choose, whereas XtreemOS does only care about identities rather than real persons.

Identity (User Identity) Identity is a set of qualities or characteristics by which an entity is recognizable as a member of a group. In XtreemOS, a user must have a user identity that distinguishes him/her from other users within the same group (a group here may refer to a node, a VO or the Grid). User identities are employed by XtreemOS software for the protection, isolation, auditing and accounting of users' access of resources, to keep system integrity and security.

Grid User Identity A grid user is identified by his X.509 certificates (proxies are actually temporary duplications of certificates). The grid user identity is independent from any VO. By possessing the certificate the grid user attests a binding of a Distinguished Name (DN) with a public key, which could be recognized by any nodes within any VOs. There should be a trust relationship established among Certificate Authorities (CAs) which issue user certificates, e.g., all are members of international PKI federations like PMA.

Local User Identity A local user is identified by his Linux/Unix account in a node.

Virtual Organization From the description of work of XtreemOS [19], "A Virtual Organization can be seen as a temporary or permanent coalition of geographically dispersed individuals, groups, organizational units or entire organizations that pool resources, capabilities and information to achieve common objectives." There usually will be legal or contractual arrangements between the entities. The resources can be physical equipment such as computing or other facilities, or other capabilities such as knowledge, information or data.

The complete definition of VO, from the technical perspective, is described in D3.5.2 [10].

Virtual Organizations can provide services themselves and thus participate as a single entity in the formation of further Virtual Organizations. This enables the creation of structures with multiple layers of value-added service provision.

A VO is created by a VO administrator and it has a guaranteed lifecyle. The information maintained in a VO refers to membership and policy information that distinguishes a VO from another. The key components of a VO are:

- an owner/administrator of the VO.
- a set of participating users in different participating domains. A user can belong to one or multiple VOs and his membership in a VO is independent to other VOs.
- a set of participating resources in different participating domains. The contribution of resources to a VO is fully controlled by domain administrators, which could then be delegated to VO administrators to assign resource access rights to grid users according to VO policies.

- a set of roles which users/resources can play in the VO.
- a set of rules/policies on resource availability and access control. Access control and use conditions (such as disk quota and memory limitation) to specify how users access resources under what kind of constraints. Up to now VO policy models of XtreemOS are not limited to a fixed one and may support fine-grained authorization based on groups, roles and capabilities as in VOMS [14], or more complicated language-based policies as in [50, 22].

For the purposes of XtreemOS, we don't model VO Goal or Workflow, though XtreemOS tools should allow these to be supported at the application level. This will typically require enforcement of policies, event notification of the completion of processes, and monitoring of exceptional events, such as jobs still executing at VO expiration. Similarly, we would not expect kernel support of contractual arrangements, but require monitoring and enforcement of policies which can be derived from contracts.

A VO and its implementation by an operating system can reside in several stages of VO lifecycle: VO Identification, VO Formation, VO Operation, VO Evolution, and VO Dissolution. In each stage a set of security threats to the overall system exist.

Since the overall security of a VO depends on its weakest component, the mechanisms for managing a VO have to ensure sufficient security properties presented at a particular VO node. Defects of traditional security mechanism can result in a lack of confidentiality, integrity, accountability and availability, e.g. by an invited VO node which does not adhere to a certain policy.

Runtime VO information Runtime VO information refers to application-generated information in a VO, which may comprise:

- Active users in a VO : Users who have started their grid sessions (see below) and they are not yet terminated.
- Running processes and active jobs in a VO: Main activities occurring during grid users' sessions, which are processes or process groups running on nodes under the control of VO execution management (job management) services.
- Resource usage tracking information: Non-repudiation information for recording resource usage by grid users for the sake of accounting and auditing.

VO Management The management work of a VO resides on two levels: VO-level (or global level) and node-level.

VO-level Management The VO-level management in XtreemOS is mainly done by SP3 (XtreemOS-G) services, which includes membership management of users and nodes that join in or leave from a VO, policy management (e.g. group and role

assignment), and runtime information management (e.g. querying active processes or jobs in a VO).

Node-level Management The Node-level management in XtreemOS is mainly done by SP2 (XtreemOS-F) services. The main responsibilities of node-level management include: translating from grid identities into local identities; granting or denying access to resources (files, services,...); checking limitations of resource usage (cpu wall time, disk quotas, memory,...); protecting and separating of resource usage by different users; logging and auditing of resource usage, etc.

Node A node is one or more machines that equipped with a single operating system image, and it belongs to a single administrative domain. There are a variety of resources in a node (e.g. CPUs, memory, files,...) which could be contributed to one or more VOs at the same time but under different policies. A node is generally identified by its network address (e.g. hostname or ip address).

Resource Various objects in a node of XtreemOS can be viewed as resources, which include files, IPC objects (shared memory, sockets, pipes) and whatever objects can be accessed by grid applications.

Proxy Credentials (Proxies) A proxy credential is a short lived pair of a new private key and certificate (called *proxy certificate*), which is signed using the user's long-term credential. A proxy has the same function in representing the user as his long-term credential. A proxy certificate could be extended as needed to accommodate attributes information obtained from one or more VO policy services, which could then be transferred to resources for making authorization decisions.

Session Similar to a traditional login session in Linux, a grid session in XtreemOS is the period of activity between a grid user logging in and logging out of the Grid, with logging-in accomplished by generating user proxies after authentication and acquiring attribute information, and logging-out done by either explicitly destroy proxies or configurable automatic expiration of proxies. During a session, a grid user can launch any applications he/she is entitled to use. All local resources allocated during a session, like local UID/GID, as well as temporary files generated during the session, will be released at the end of the session. A detailed description of how sessions are managed will be given in the following sections.

2.3.2 Overview of VO Support

As described in the section above, VO support functionalities in XtreemOS are realized by the cooperative activities of VO-level and node-level management services. Particualrily, the main objectives of VO support in WP 2.1 are:

- to faclitate the administration tasks for a single PC node to join or leave VOs

- to enforce VO policies locally with system-level isolation of multiplexed VO accesses to the same node
- to significantly increase the usability of shared resources on local nodes for VO users

The key challenge here is to reach a harmonisation between VO-level policies and local policies on nodes which depend on autonomic domain administrators. On one hand, the enforcement of multiple VO security policies should be differentiated, on the other hand, this kind of enforcement should not be conflicting with any local policy of nodes and it will not impaire the usability of resources for grid users.
Basically, main tasks done in WP2.1 for VO support are:

- User Identity Transferring
  The task is to develop an account-based mechanism for the smooth transferring from global grid identities to local Linux accounts. During a session, Linux-XOS must maintain a strict mapping table between grid user credentials and the local Linux accounts used for running application processes, for the sake of access control, logging, accounting and revoking of resources.
- Resource Management and Access Control
  The task is to develop resource management mechanisms compatible with local Linux Discretionary Access Control (DAC) model while supporting required access control specified in VO policies, which, to the contrary, generally adopt the Mandatory Access Control (MAC) model. Integrating a global MAC model inside the Linux kernel could require heavy modifications to the kernel and may incur non-trivial penalty of performance. Providing the same effect of MAC on basis of original Linux DAC model is complicate without the support of kernel-level isolation techniques.

A user can run processes on a node of XtreemOS as a

- Local user
  The user accesses a node as a local user who is identified by his local account in the node, which is the same as in traditional Linux. As a local user, one cannot access grid services.
- Grid user
  The user accesses a node as a global user, (a) The user must be identified by the Global Identity, his X.509 certificate which presents a Distinguished Name for the user, and (b) the access must be limited under a VO context. That is to say, the user must be a member of the VO and the VO policy is in charge of the user's authorization in the node.

Once a Grid user process is running on a XtreemOS node, it can

- Open files/pipes/sockets
  Access rights to the file are checked by the supporting filesystem. Local filesystems use Linux DAC/ACL model (local UID/GID). Grid file systems base authorization on user credentials (certificate/attribute/proxy/ticket) stored in the process context (keyring in Linux). Linux-XOS must provide access control for all files stored in local filesystems.

- Communicate with a grid service
  Communication with grid services is handled by grid services libraries. Commication rights to grid services must be checked using the user credentials. Grid service library extracts user credentials from the process keyring through some API.

- Signal processes
  Signalling local procces of a Linux-XOS node is possible and is under control of the DAC model. XtreemOS requirements request for a global management of processes of a grid session or of a VO. As long as XtreemOS nodes are unaware of the global process name space, signalling a global process is possible only through a grid service API.

User credentials are usually limited in time and must be renewed periodically. Some VO models give the possibility to add new attributes to existing credentials.

- Credential Renewal
  Upgrading existing credential (renewal) must be possible in a secure way.

- Interoperability of Credentials
  Interoperability itself is a big challenge for the heterogeneous systems. But the interoperability of the security related information is crucial for the overall security of a system. It will be unrealistic to anticipate that all the VO nodes will have the same kind of credentials (certificates, tickets, etc.). VO should support the secure interoperability of the credentials of its various nodes.

XtreemOS nodes are in charge of running user applications on behalf of virtual organizations. The appartenance of a node to a VO can be managed dynamically. XtreemOS must provide means to report all accounting, logging and auditing informations to virtual organizations before a resource is removed from a VO.

2.3.3 Overview of the Basic Implementation

Session Management

A grid session on a Linux-XOS node covers all activities on beneath the same credentials. A grid session starts with the acceptance of the user crendential. When the session is terminated, no more activity can exist on behalf this credential. Opening

a new session on a grid node takes the three classical steps: authentication, authorisation and session creation. These three steps can be implemented in XtreemOS through PAM plugins, or using a specific PAM-like service.

Authentication the credentials presented with the request (ticket, proxy certificate, ...) are checked and validated: user identity from some certificate authority, attribute certificates from VOs. Depending on the VO model, this phase can necessitate transactions with external grid services: MyProxy for user identification, user attributes from Shibboleth, etc.

Authorization after the request has been fully authenticated, further authorization checks can be taken, for instance that access to this node is not denied for this user.

Session creation the last step after authentication and authorization is the creation of a user context. This step involves the selection of a local UID/GID for the session, the initialisation of the user environment (from the job description document), creating a scratch homedir or mapping the user homedir from a grid file system, starting some auditing/logging/monitoring/management service, storing the proxy certificate in the session context (keyring), dropping some capabilities and running the user request.

A session on a grid node can be limited to running a single process (simple session). It is also possible to interact with the session through the associated management service, for instance to start new processes (composite session). XtreemOS API must provide means to run simple application –a single request for the whole execution– and to run complex applications –a session is first opened and then multiple execution requests can be handled in this session–. Session can be interactive or not. To facilitate interaction with applications using GUI interfaces, XtreemOS sessions must provide secure X11 forwarding as in SSH.

The session is terminated when

- the application of a simple session is terminated,
- the management service received an end-of-session request,
- the proxy certificate is no more valid.

Linux PAM

Linux-PAM is a system of libraries that handle the authentication tasks of applications (services) on the system. The library provides a stable general interface API that privilege granting programs (such as login and su) defer to perform standard authentication tasks (from Linux-PAM manpage).

The principal feature of the PAM approach is that the nature of the authentication is dynamically configurable. The system administrator is free to choose

how individual service-providing applications will authenticate users using simple configuration files.

In XtreemOS, a domain system administrator installs the plugins corresponding to the VOs having access to the local nodes. The system is extensible: new plugins corresponding to new VO models (or credentials) can be developed and installed dynamically. Virtual organizations managed using different VO models can cohabit on the same node.

Local UID Management

In order to run processes on a grid node, a local UID/GID must be allocated for each session. This UID/GID can be static (the user owns a local account on the node) or dynamic (the user is unknown). A dynamic UID/GID is returned to the free UID/GID pool at the end of the session once all corresponding processes have terminated and once all local objects (files, etc.) have been deleted.

The possession of a local account can be managed in a kind of gridmapfile (a configuration file listing user Distinguished Names and the local accounts) and analysed by a session PAM plugin. It should also be possible to specify the local account identification using a trusted (by the local domain) attribute of the user proxy certificate.

Session Interaction

Complex Grid applications can be deployed across multiple grid nodes. As long as two grid sessions are opened on two different grid nodes, the only means of communication is through *grid services*. The use case requirements do not request for specific handling of the case where two grid sessions from the same user run on the same node. In the basic version of XtreemOS, in order to avoid file naming clashes, different concurent sessions from the same user will be allocated different UID/GID unless a local account is explicitly specified in the proxy certificate.

Access control

Access control to an object in XtreemOS depends on whether it is a kernel object or a grid object. Kernel objects are local files, local processes, local shared segments, ... on a grid node. Applications access these objects using native Linux API: open/read/write for files, signal for processes, ... In order to avoid deep modifications to the Linux kernel, access control to kernel objects should use the native DAC/ACL system of Linux in the XtreemOS basic version.

Access control to grid objects will be managed inside grid services (grid file system, etc.).

It is possible to “cache” grid objects inside the kernel in order to improve efficiency. This is the case of local grid file replicas. The initial open request on a grid file is handled by some grid service (the grid file system) which checks for access

rights using the user proxy certificate. Once access has been granted, the filesystem service can create (or reuse) a local replica of the file on a local file system and then forward the open request to this local file system. Access rights to this local replica must be defined using native DAC/ACL in such a way that the initial open request succeeds. But these access rights must not grant more rights than the original grid file for other users.

Access rights to replicas are removed by the grid file system service when the file is closed.

In the case where many concurrent sessions open the same grid file, they use the same local replica. Linux ACLs appears to be the most appropriate tool for handling this case.

XtreemOS files

A XtreemOS-F node should handle four kinds of files:

Grid file access rights to these files are controlled by the grid filesystem service using the proxy certificates. Grid files are distributed across the XtreemFS Object Store Devices (OSDs), their metadata being provided by the XtreemFS Metadata and Replica Catalogue (MRC). For the node it is completely transparent whether it accesses a primary file instance or a replica. XtreemFS will implement mechanisms to help users to influence replication policy and replica location. If a node runs an OSD service, the user will be able to request that a replica is located to this particular node.

Inter-domain file NFSV4, for instance: NFSV4 has its own access control mechanism with a kind of user/domain id. NFSV4 implements a named mapper daemon which is in charge of `UID/GID <--> user/domain` translations. It is possible to configure NFSV4 on XtreemOS nodes in the following way: the NFSV4 `user/domain` id of the user is an attribute (trusted?) of the user proxy and is added to the NFSV4 named mapper at session beginning and removed at session end by the PAM session plugin.

Persistent file If a grid user has a permanent mapping to local UID/GIDs, he/she can possess local persistent files on the node. Global to local UID/GID mappings can be made sticky on a node, for example in the case that a user owns a local UNIX account on the machine. It should be possible for the user to express this situation by using some attribute (trusted by the local domain) in his proxy. At session begin, his application is forked with his permanent local UID/GID. The user is then free to manipulate his access rights. Access rights to such files are controlled by UID/GID. A similar case can happen if the user wants to access files owned by another local user.

Temporary file the grid user can create temporary files in some local filesystem (/tmp or other). The number of directories where some user can create temporary files should be limited in order to limit the cost of tracking these files

at the end of the session. All these files are deleted at the end of the session. Access control is through UID/GID.

2.4 Modifications to Linux

Approach Outline

Virtual Organizations (VO) support in Linux requires the control, mapping, allocation, monitoring and enforcement of global, grid and VO visible resources onto single Linux nodes. Linux is unaware of the global grid entities (eg. global user ID, VO ID) therefore one of the tasks of this work package is to add mechanisms for recognizing, controlling and enforcing their usage on the Linux machines. The approach for adding these mechanisms could be chosen between two extremes:

- Implement global grid entities into Linux and add mechanisms for controlling them.
- Map global entities to local ones already available in Linux, use already available mechanisms for controlling and enforcing them.

For reasons explained in Section 2.3 we chose the second approach and will use existing infrastructure in the Linux kernel:

- Global grid entities will be mapped to local ones (like user and group IDs (UID/GID)). Existing mechanisms for local entities will be used to enforce resource control.
- The kernel key retention service will be used for storing user private keys, certificates and proxies associated with user processes and sessions in kernel space. This mechanism will avoid the need for kernel changes for transporting grid-related information in process context.
- The existing mechanisms of security enhancement for Linux, such as Linux Security Module (LSM) framework [54] will be investigated to refine the access control and resource enforcement mechanisms. It is discussed later within this section.

The chosen approach is minimal with respect to core kernel code changes and tries to keep required kernel changes localized in dynamically loadable kernel modules. This reduces the pressure to get VO related changes accepted by the kernel developer community. The interaction between kernel space and user space daemons will be done through existing APIs. In addition, the use of existing kernel infrastructure like the key retention service and LSM provides opportunities to build up good relationships to the authors and maintainers of these components.

2.4.1 Fundamental Modifications to Linux

As the main approach adopted is basically account mapping between grid user identities and local user identities, modifications to Linux mainly reside at system service level rather than kernel level. As described above, the following extensions are added to facilitate the use of VO functionalities by users:

- A new PAM module will be developed to take over the initialization of grid sessions. After authenticating users and getting authorization information from higher-level VO services, this PAM module will do account mapping and translate VO-level policies into POSIX-compliant local policies, i.e. UID/GID, ACL and POSIX Capabilities. This PAM module itself is designed as a pluggable framework to fit with various high-level VO models. For example, authentication plugins are capable of authenticating users with respect to their PKI certificates, or from a MyProxy repository, or by interacting with federated identity providers like Shibboleth. To obtain attributes information for users, authorization plugins could be developed to access VOMS servers or role-based access control frameworks like Permis. The plugin architecture of this PAM module makes node-level management mechanisms independent from higher-level VO frameworks.
- Besides the PAM module guarding the login process, there will be runtime services that monitor and control users' activities during their grid sessions. The responsibilities of these services include checking and adjusting resource limits of processes in a VO context (e.g. by set_rlimit()), logging resource usage of processes, and providing error or debug information feedback to users.
- System services and utilities such as SSH could be extended to allow grid users to use remote nodes interactively without the need of explicitly creating traditional user accounts. Once login, grid users with identities like certificates could have a shell access to remote nodes in a VO. It may favor the requirements from advanced users who wish to develop and debug applications on grid nodes rather than being limited to submit batch jobs. This modification allows for a mimic approach of using the Grid as same as using traditional high performance computers.

2.4.2 Open Issues

The implementation of the basic XtreemOS-F version requires few and very local modifications to the Linux kernel. This strategy leaves some open issues which will have to be solved during the course of the project:

Session management In the basic version of XtreemOS-F, grid sessions will be managed using Unix sessions. As long as all processes created on some node belong to the same session, it is possible to globally control these processes (signal,

kill, wait until all processes are terminated). But, in Unix, a process can dynamically decide to leave the session and to create a new session. Such processes are out of control of the grid session management. With limited modifications to the kernel, it is possible to control session creation, using capabilities, LSM [54] or SELinux [46]. In a next step, it is necessary to evaluate if this capacity to fork new sessions is really needed for grid applications in XtreemOS.

Accounting VO management requests for precise resource usage and accounting. Some use cases request for near real-time feed-back of resource consumption. Linux provides weak global monitoring of multi-process resource usage.

2.4.3 Initial Thoughts on Advanced Approaches for VO Support in Linux

The account mapping approach is a simple but efficient way to isolate accesses to local nodes by different grid users. However, in this approach, it is complicate to maintain the mapping table of grid credentials to local Linux credentials and VO policies to local Linux capabilities, especially when VOs are dynamically changed. Our initial thoughts on improving the account mapping approach are:

- Leveraging the Mandatory Access Control (MAC) support in Linux. MAC support mechanisms, such as LSM [54], which are taken as security enhancement to Linux, have been incorporated into the latest kernel. LSM provides a collection of hooks in kernel which make it possible for developers to custom special security check policies for various objects access. LSM enables a direct enforcement of VO policies in local nodes without the translation of them into local capabilities like file permission bits and ACLs, which could facilitate the management of dynamic change of VO policies.

- Leveraging operating system-level virtualization techniques in Linux. Operating system-level virtualization has its advantages over emulation or paravirtualization based approaches in terms of low overhead for accommodating hundreds of virtual servers and low setup costs for instantiating virtual servers on one physical machine. It provides another way for separation and isolation of different VO accesses to the same local node. The challenge here is to prevent misbehaving virtual servers crudely exhausting resources from impacting good ones.

To make advanced approaches for VO support a practical solution, further experiments need to be conducted to make sure they are easy enough to use without impairing the usability of the whole system, and they will not incur heavy additional overhead. Whether they will be adopted in future version of Linux-XOS is still under investigation.

2.5 Comparison with Other Approaches

In this section, we present a preliminary comparison of VO support functionalities between XtreemOS and existing approaches. This study is conducted from the following aspects: usability, security, overhead and performance.

Usability

The cost of administering and operating a VO (e.g., adding or removing nodes, changing access policy, authenticating and authorizing users) should be minimized to a bounded value rather than simply increase with the number of users and resources participating in the VO. Moreover, the dynamicity of users and resource usage needs to be handled in a flexible way. Users and resources are often autonomous, not subject to the control of a centralized entity. A user or a resource can join or leave a VO at any time. Such cases could bring heavy management burdens to administrators, which must be alleviated by the newly designed mechanisms.

In contrast to a toolkit approach such as Globus, where there are two separate but highly inter-dependent complex entities to be managed in tandem (the underlying OS and the middleware), the XtreemOS project adopts an approach in which the standard operating system running on a machine is a Grid OS, that is to say, the operating system is fully Grid-enabled. Once the XtreemOS system has been installed on a machine, this machine is ready to participate in a VO with no need to install additional system software. Modifications to Linux to natively support VOs are done with a careful design to keep backward compatibility while providing build-in VO management interfaces that are as secure and simple to use as possible. System services and utilities such as login and shell programs, together with libraries, are extended in a modular approach so as to favor VO-level resource sharing requirements while keeping maximal transparency to users.

Security

OS kernel-based mechanisms have been fairly conclusively proved to be effective at providing application isolation, which allows resources to be strongly protected from compromise by malicious or malfunctioning programs, a level of security that is generally hard to achieve at the middleware or application level. By integrating the necessary mechanisms for operating a Grid node and for managing VOs into the kernel, a similar level of security can be provided for Grid applications and VOs.

- Better protection of proxies
  Protection of proxies in grid middleware is important as they are stored in temporary files, however, due to the nature that the middleware in itself are user space applications running on top of an OS, local malicious or malfunctioning programs may intrude and affect the grid applications by hacking

proxy files. XtreemOS could provide better protection of proxies by storing them in kernel space.

- Better enforcement of VO policy
  The XtreemOS approach also addresses the "secure perimeter" weakness by eliminating (or at least substantially reducing) the gap between the security enforcement point (traditionally in the Grid middleware) and the action that has to be protected (typically a kernel action such as disk access or job launching). By extending current OS kernels with native support for VO management, XtreemOS could empower grid applications with more fine-grained access control of cross-domain resources than grid middleware solutions, as the latter still rely on a coarse mapping of grid user identities to traditional local accounts for enforcing authorization, or worse, rely on gatekeepers to ensure unauthorized actions are not requested.

Overhead and performance

While well-designed Grid middleware with a layered architecture design can increase conceptual efficiency by raising the abstraction level, this is often at a cost of execution efficiency, particularly in the interaction overheads that dominate small-grained or highly dynamic grid systems. The XtreemOS project will attempt to collapse multiple conceptual layers into one efficient OS-level implementation, to eliminate this structure/performance trade-off. Not only would this reduce interaction overheads, but for those computation intensive grid applications with stringent performance requirements, it should generate more opportunities for performance tuning in terms of the information available for optimization and the scope of optimizations.

- Account mapping approach
  The account mapping approach, which extends traditional Linux DAC model to support VO access control, has the minimized overhead for grid applications. The key of this approach is that the translation from VO policy to local ACLs will not happen at runtime so that any access control check for VO resource access is done just like that for traditional Linux file access.

- Advanced approaches
  The advanced approaches leveraging MAC support or virtualization support in Linux, although still under investigation, could be an alternative or even enhancement to provide better VO management. An initial thought on the MAC approach is to utilize a special kernel module on top of the LSM framework to enforce VO policies and do access control checks. Much overhead can be avoided if well-designed policy cache mechanisms are utilized, for example similar caching mechanisms like AVC in Flask architecture [46].

Chapter 3

Checkpointing Linux Processes

In the context of XtreemOS, applications are composed of application units running on different grid nodes. An application unit is then defined as a collection of processes under the control of one operating system instance (*ie* a grid node), either Linux-SSI or Linux-XOS. These processes could be multi-threaded.

Due to the dynamic nature of virtual organizations, an application unit running on a grid node may need to be moved to another node during its execution. In the same way, an application may need to restart one of its application units that has experienced failure of the node it was running on. Therefore, Linux-XOS should implement methods and interfaces to checkpoint and restart applications. The description of work (Annex 1) [19] gives the following description of the expected properties of checkpoint/restart mechanisms that should be included in Linux-XOS:

1. It will be for instance possible to checkpoint a multi-threaded application (OpenMP, POSIX threads), a MPI application or any socket based application using either the programming or the command line checkpointing interface.

2. An application unit running in the context of a VO should be able to be restarted on a different node (same architecture) within the given VO.

3. the checkpointing mechanisms should be efficient and generic enough to cope with the different communication paradigms used by the applications (within an application unit and between application units of a distributed application). A grid-level application unit checkpoint has to be independent from the operating system instance of its execution node.

4. Checkpoint/restart mechanisms for applications composed of a single process[1] will be designed and implemented in the Linux kernel. Application- and system-initiated checkpoints will be implemented. Checkpoint/restart

[1] application unit in the original document

mechanisms will be implemented for both multi-threaded processes[2] and application units composed of a set of communicating processes.

This section describes how we intend to meet the stated objectives for checkpoint/restart mechanisms in Linux-XOS. Specific application requirements are first described in Section 3.1. In Section 3.2, we present the state of the art in Linux Checkpointing and discuss the difficult points. In Section 3.3, XtreemOS's checkpointing architecture is presented before the specifications of checkpointing in Linux-XOS in Subsection 3.4.

3.1 Application Requirements

The following requirements are extracted from the consolidated requirements documents produced by WP4.2. The first list is the list of requirements which are labelled as obligatory, with for each requirement the original definition, comments by WP4.2 to clarify the definition and comments by WP2.1 to rephrase the definition in the vocabulary used by this document.

3.1.1 Requirements labelled as obligatory

- **R28: XtreemOS must support automatic failure detection, checkpointing and restart.**

  > "XtreemOS must provide automatic failure detection, checkpointing and restart at the system level preferably with no need to modify the application.
  >
  > Restarts should be done from the last checkpoint, unless the user specifically requests the use of an older checkpoint. Alternatively, if the application fails a few times in a row from the last checkpoint, an older checkpoint can be tried automatically."

  Failure is understood here as failure of a node or of one of the node's components (disk, network access) leading to the failure of process.

- **R30: XtreemOS must notify the application of checkpointing and restart.**

  > "XtreemOS has to notify the running application prior to checkpointing and interruption such that the application has the opportunity to react appropriately (e.g. store data, close open files, send messages). Furthermore, XtreemOS must notify a restarted application of the changed execution environment, including but not limited to IP addresses, hostnames and PIDs."

[2] id.

If the application has to react, it must be modified, which seems contradictory to R28. It is therefore understood that this requirement could be reformulated as: the application should be notified that a checkpoint is being taken so that it can be optimised for quicker checkpointing by taking appropriate actions, and should be notified of restart so that it can take into account being moved to an other node.

- **R31: XtreemOS has to support various ways of checkpoint initiation.**

  "It is required that XtreemOS provides the mechanisms for creating and storing sequences of checkpoints. The checkpointing mechanism needs to be configurable to support:
  - Checkpointing initiated by the application independent from the OS.
  - Checkpointing initiated by the application to stable storage provided by the OS.
  - Checkpointing initiated by the OS at application's request.
  - Automatic checkpointing initiated by the OS (with and without notification).

  A respective API has to be provided."

- **R32: Checkpointing must be fast enough that the required checkpointing frequency will not represent a significant load to the system.**

  "Restart from a saved checkpoint must not take more than the time between successive checkpoints, i.e. for one checkpoint each 20 minutes, at most 20 minutes should pass from failure to the moment when the application is up and running again.

  Quantification: One application has extremely high demands: at least one checkpoint per 30 seconds. The estimated amount of working memory per node for this application is 512 MB, but the size of the data requiring checkpointing is much smaller. The checkpointing frequency must be high for online application to allow restarting quickly after failure and without loosing much data; otherwise the user experience will suffer.

  Other applications demand at least one checkpoint per hour. Their memory requirements are specified as up to a few GB."

- **R34: XtreemOS allows customisation of checkpointing and restart**

  "It is required that XtreemOS allows the applications to specify on which nodes a checkpointed application will be restarted. This is important e.g. for applications that require user interaction.

  Restart from last and older checkpoints must be customizable to support:

  - automatic restart.
  - restart with user-interaction.

  XtreemOS must allow the user to specify how often (or when) checkpoints are created. XtreemOS must allow to activate/deactivate checkpointing and automatic restart during application runtime with no need to reboot nodes."

- **R35: Information on the process state that must be saved/restored during checkpointing/restart.**

  "Checkpointing and restart must save/restore the following information on the process state:
  - Threads.
  - IPC.
  - Network communication (in particular open MPI communication).
  - Open files (contents must be saved).

  Optionally, linked libraries should also be saved. The automatic restart with the same threads may use the mechanisms already implemented in Kerrighed. The involved restart can perhaps use similar mechanisms to those that will be used for normal application startup, with a flag signalizing that this is a restart. Saving contents of large open files is not realistic. If the files are not open for very long, taking the checkpoint can be delayed automatically for a few seconds, hoping that there will be a moment when no large file will be open. Otherwise, the applications will have to cooperate on this issue."

3.1.2 Requirements labelled as optional

This second list lists the requirement labelled as optional.

- **R29: Restart should mimic the original environment.**

  "XtreemOS should provide the illusion of the original environment considering in particular for IP addresses, hostnames and PID numbers. Accordingly, a mechanism for handling virtual IP addresses, hostnames and PIDs should be available."

- **R33: Checkpointing/restart should be implemented as kernel module.**

  "Checkpointing and restart must be implemented on OS level preferably as kernel module."

  The motivations for this requirement aren't very clear, but it is assumed that the underlying requirement is that the application should not be recompiled to use checkpoint/restart mechanisms.

3.2 State of the Art

3.2.1 Introduction

Checkpointing can be defined as taking a snapshot of the current state of a process (a checkpoint) so that the process can be restarted from that state at a later time. We therefore face four challenges when dealing with checkpoint/restart mechanisms.

The first challenge is defining the state of a process. Indeed, a process' state is seldom contained only in the processor and memory context of that process. Some of its state is linked to the environment it is running in, including the operating system it is running on. Indeed, a process may be aware of the hostname of the node it is using, as well as of its network address. It will probably have open files or open connections, and links to the VO it is running under as well as some form of user interface. Between the moment a process has been checkpointed and the moment it is restarted, the members of its VO may have changed and other applications, including itself during the period between the time at which the checkpoint was taken and the failure of the process, may have changed the state of files or databases the process was using. Moreover, its user's working environment will have changed.

Taking a snapshot of the state of the VO, the file system, any connected databases, all connected processes as well as its user's environment as part of the state of a process is neither realistic nor desirable. Therefore, particular attention has to be taken in defining what is part of the state of a process and what is part of the state of its environment. We need to define process boundaries.

The second challenge can be seen more as a technical challenge: how does one take the snapshot of the process' state. An initial approach could be the one taken by any computer user that saves the document he is working on before leaving his application. His work can be restored (restarted) by starting his application again and re-opening the document he was working on. This approach however does not help the administrator of a system that needs to take checkpoints of applications before he takes a node down for maintenance. Taking a snapshot of a process should therefore be possible for all processes of an operating system, or at least for all processes started in a checkpointing context by the operating system. In this document, we will focus on the different approaches taken by projects implementing a checkpoint/restart mechanism for Linux.

The third challenge is managing the snapshots that have been taken. In the context of autonomic computation, it has been suggested that the operating system transparently takes checkpoints, manages the taken checkpoints to destroy them once the checkpointed process has successfully completed. In the event of a failure of the process, the operating system would automatically select one of the checkpoints and restart the application. In this view of checkpointing, snapshots are not accessible or known to the user, and the system keeps complete control of their location, usage and usefullness. This property is difficult to maintain if snapshots are used to migrate processes from one node to another, especially in Grid contexts.

Moreover successful completion of a process does not imply successful completion of the application it was part of, and it could therefore be possible that snapshots of that process are still necessary to restart the whole application in the event that an other process was using a node that has failed. Therefore, most checkpoint/restart systems available for Linux either store the snapshot to a file, or for more evolved systems, to a file descriptor, which enables snapshots to be transparently sent to another node, which is required to be able to restart the process should the local node fail.

Finally, because of requirement R35 mentioning the need for mechanisms to checkpoint processes involved in an application implemented using MPI, this state of the art will also take into account checkpointing strategies for MPI programs, for the case these strategies involve a system checkpointer and are not completely hidden in the MPI library.

3.2.2 Defining Application Unit Boundaries

Defining what the state of a process is, is not a trivial task. Processes with no interaction with their environment are rare. Most processes at least produce or consume data in some form or another. These interactions can be classified in the following way:

- interactions with persistent storage: reading, or writing files.
- interactions with other applications: interaction with a window manager, a database server or a grid service.
- interactions with other processes of their application: this can be through network connections, shared files, pipes, IPCs or process relationships (process group, parents/child, session).
- interactions with specific devices, such as graphic cards or specialised equipment.

Moreover, these interactions may use information found in the local environment, such as node name, network address and process identifiers to enable.

Interactions with persistant storage

The BLCR [33] project identifies five possible behaviors for handling an application's files, and therefore five views of an applications boundaries as far as opened files are concerned:

Simple reopen In this model, no information about the file is saved in the checkpoint, except for the name of the file, and the current file offset at the time the checkpoint was taken. At restart time, the file is expected to exist in the same file system location, and it is reopened and seeked to the checkpoint-time file offset.

Checksum and reopen This model is identical to simple reopen, except that a checksum of the file is also stored at checkpoint time. The checksum is reapplied to the file at restart time, and if the file has changed, the restart fails.

Truncate This mode is really a modification that can be applied to the first two approaches. The size of the file is recorded at checkpoint time, and at restart the file is truncated to its checkpoint-time length. In the checksum case, this is done before the checksum is reapplied. Truncation provides a primitive rollback mechanism for files that are written to in a continuous stream, which is the access pattern used by many scientific applications. It allows such applications to be restarted in an idempotent fashion multiple times and still get a coherent output file.

Backup and restore In this model, a full copy of the file is made at checkpoint time, and stored as part of the checkpoint. When the application is restarted, the stored copy is restored onto the file system, overwriting any existing version.

Backup and anonymize As with the previous model, a copy of the file is made during checkpointing and restored at restart, but the restored file is only visible to the restarted application. This mode may be useful for handling a fairly common idiom in Unix programming, in which a file is opened by an application then deleted from the file system, remaining visible only to the application and existing only while the application keeps its file handle open.

They estimate that each of these behaviors can have a use in some contexts. Therefore they suggest that the user chooses the behaviour that he wants at checkpoint time. Ideally, the user should be able to specify the behaviour on a per-file basis.

Implementation of these different strategies can be eased if the file is stored on a file system that supports snapshots. Nevertheless, if the file has been changed by another application, and not by the checkpointed application after the checkpoint was taken, it implies that in one way or another the application is interacting with another application, and the best strategy is probably to prevent the restart.

For the time being, most checkpoint/restart systems for Linux only implement the simple reopen strategy, with the notable exception of UCLiK [26].

Interactions with other Applications

If part of the checkpointed process' state is owned by another application, ie the process is interacting with another application, there is no generic means of extracting and thus saving part of the state of the checkpointed process. Indeed the other application may not even be running XtreemOS, and therefore XtreemOS's checkpoint/restart mechanism won't be able to do anything.

Interactions with other Processes

Some processes are part of a wider scale application, and could be interacting with the other applications units of that application through different means, such as network connections, shared files, pipes, IPCs or process relationships. In this context, it makes sense to be able to checkpoint the application by checkpointing all application units and processes as well as the objects used for interaction between application units. BLCR (as described in [33]) has therefore designed their checkpoint/restart mechanisms to be used at the application level, using the MPI library as a testbed and their effort is examined in Subsection 3.2.5. The idea here is that given the variety of the technical details implementing the relationship between an application and the different processes it is composed of, the checkpoint/restart system of a process cannot direct the checkpointing of the whole application, especially if this application spawns different administrative domains as in a Grid. Nevertheless, the checkpoint/restart system can be designed to enable application writers, or more realistically writers of libraries for parallel or distributed computing, to add checkpointing capability to their code, based on the checkpointing facility offered by the operating system.

Interactions with Specific Devices

In this case, the device driver needs to be written in such a way that its state can be extracted and restored. This can be provided in the general context of the Software Suspend hibernation mechanism used in the official Linux kernel source code. Nevertheless, Software suspend enables saving the state of a device for all of its uses by the operating system, and not by a single process. This is a direction that has very little been explored.

State Linked to the Local Environment

To minimise modifications that need to be made to applications using checkpoint/restart mechanisms, some projects attempt to run applications in a virtualised context whose restoration can be guarantied upon restart. This virtualised context virtualises all information that an application unit could attempt to use to interact with its environment. This is the case for the MCR [16] project, which requires that all checkpointable application be run using a specific command (`mcr-execute`), for ZAP [41] using PODs or for Crak [30]. For MCR and ZAP, the pids as seen by the processes of an application unit are virtual pids only meaningful in the context of the container the application unit is running in. In these projects, some kind of private space for network addresses and hostnames are used to enable transparent migration of applications. This virtualisation can be useful at process level (in case a process uses a temporary file whose name contains its process id for example) or at the application unit level for all processes of an application running on the same node. But it does not solve the problems that arise when this information is used to interact with other applications, for example if the IPv4 network address has been

given to another application, it would be quite difficult for this other application to recontact the originating application once it has been restarted on another node.

3.2.3 Taking a Snapshot in Linux

The different mechanisms used by the different checkpointing libraries available for Linux can coarsely be classified between user-level approaches and kernel level approaches.

With user-level approaches, the application needs to be compiled or linked against the checkpointing library either at compile time or at run-time (using the LD_PRELOAD mechanism). This is the case for ckpt[37], used in Condor or libckpt[20]. In the first case, a checkpoint is taken in the context of a signal handler, as the checkpoint is taken when the process receives a signal. In the second, a checkpoint is taken in the context of a thread of the application, either at another thread's request or upon expiration of a timer. The main drawbacks of these approaches is that extracting state information from the kernel is costly and the alternative, maintaining the needed information in user space by intercepting system calls is fragile.

With kernel level approaches, many more strategies can be implemented. These strategies differ in the way they enable the application to react or participate in the checkpoint/restart mechanism. BLCR for instance implements a callback mechanism that is used before and after the checkpoint is taken to tailor the checkpoint/restart mechanisms to the need of the application. The callbacks are either executed in signal handler context (in this case one of the realtime signals is used) or in thread context. In signal handler context, the process is stopped, therefore no concurrence between the process and the signal handler needs to be taken care of, except maybe in multi-threaded applications. But signal context is very restrictive in term of the type of operations that can be performed. On the other hand, thread context, where a specific thread handles callback does not require that the application be stopped (it is not the case with BLCR). Therefore, concurrent accesses must be handled as suggested in [33].

[17] suggests implementing checkpointing by creating a copy of the process using the fork system call, and then saving the state of that copy. This would only work if the process is composed on only one thread, but enables very fast checkpoints from the point of view of the checkpointed process as it leverages the copy-on-write mechanisms available in Linux.

An other optimisation based on the copy-on-write principles is to take incremental snapshots of an application. In this case, only one in n snapshots is complete, and all intermediate snapshots only take snapshots of what has changed. Detecting changes can be done at the system level as in TICK [28] using a combination of book-keeping and harware support.

The different implementations also differ in the interface they offer for checkpoint initiation (system call or ioctl) and on the way the checkpoint/restart mechanisms are integrated into a running kernel. Usage of a kernel module, the preferred

path for inclusion in an existing system because it is far less intrusive, has been adopted by a number of projects such as CHPOX, UCLiK, CRAK and to some extend BLCR, as this latter project relies on two kernel patches: VMADump and a distributor patch to export the `do_fork` system call to modules.

3.2.4 Managing Snapshots

In the context of checkpoint/restart systems for Linux, very little has been done as far as checkpoint management is concerned. All projects examined store the snapshots on file, either directly or by using the file descriptor that they receive as a parameter at checkpoint initiation. This last implementation paves the way for checkpoints that are stored remotly if the implementation does not use `seek`.

3.2.5 Checkpointing an MPI Application

The MPI standard does not specify any fault tolerant behavior. However some MPI libraries provide support for fault tolerance. They are different approaches to make a MPI application fault tolerant. Using checkpoint/restart mechanisms is a solution.

To checkpoint a MPI application, it is possible to use a single process checkpoint/restart system and to add a coordinated or uncoordinated checkpoint/restart protocol to be able to create a consistent checkpoint of the entire application. Some implementations of this kind of solutions like Starfish, MPICH-V and CoCheck, are tightly coupled with a specific checkpoint/restart system. But other implementation are trying to have a more generic approach to be able to integrate different checkpoint/restart systems. LAM/MPI [45] and Open MPI [27] are developed in this way.

LAM/MPI

The design of LAM/MPI is based on a component framework, the System Service Interface (SSI). Each SSI type provides a single service and can have one or more selectable instance. The CR SSI is the interface for the checkpoint/restart modules. Two CR modules have been implemented: The BLCR module which is based on the BLCR checkpoint/restart mechanism and a self module that will invoke user-defined functions to save and restore checkpoints. One important point of LAM/MPI is that it requires that the checkpoint/restart system provides a notification to *mpirun* in order to initiate the checkpointing of a parallel job. It means that with BLCR it is the *Cr_checkpoint* utility of BLCR that is invoked to initiate the checkpointing. LAM/MPI defines *enable_checkpoint* and *disable_checkpoint* functions. They enable the definition of critical sections where checkpoints can not be taken, during MPI_INIT for example.

Open MPI

The work on checkpoint/restart with Open MPI extends the work of LAM/MPI. The requirements on the checkpoint/restart system are almost the same. But the checkpoint/restart system does not need to provide application level notifications when checkpoints occur. When the checkpoint/restart system has completed the checkpoint of a process, it must provide Open MPI with a reference to the checkpoint image generated.

Summary

The checkpoint/restart services of LAM/MPI and Open MPI are independent of the single process checkpoint/restart system used to checkpoint each MPI process. LAM/MPI only supports a coordinated checkpoint/restart protocol whereas Open MPI also supports an uncoordinated one. The requirements on the checkpoint/restart system are very limited:

- To be able to save the entire state of a single process. There is no need to preserve shared memory regions or socket connections and "in flight" messages. This aspect of the state of the application is preserved by the MPI library.
- To provide functions to define critical sections.
- To provide a notification to *mpirun* in order to initiate the checkpoint (only for LAM/MPI).
- To provide a reference to the checkpoint image after a save (for Open MPI).

3.3 Overview of Application Checkpointing in XtreemOS

This section is also inserted in D2.2.1 deliverable for the sake of specification consistency.

Based on the state-of-the-art, application checkpointing in XtreemOS involves three levels of checkpointer:

1. the kernel checkpointer, providing basic functionality to take a snapshot of a process.
2. the system checkpointer, providing checkpoint management at the application unit level, *ie* automatic checkpointing and snapshot management.
3. the grid checkpointer, providing checkpoint/restart facilities at the application level.

3.3.1 The Kernel Checkpointer

The kernel checkpointer offers a very basic checkpointing interface that enables

- checkpointing of a process,
- notification to the checkpointed process that it is about to be checkpointed,
- registration of callbacks from an application to tailor checkpointing to the application's needs,
- enabling and disabling of checkpoint from the application if it is written in a checkpoint aware way.

The callbacks are a means for the process to extend the boundaries of a checkpoint as made by the kernel level checkpointer.

3.3.2 The System Checkpointer

The system checkpointer is an OS service that manages checkpointing for an application unit. It registers checkpointing strategies and implements them.

- It will use resources given to it to call the kernel checkpointer or request those ressources on behalf of the calling process.
- It implements periodic checkpointing.
- It implements staged checkpoints.
- It implements checkpoint garbage collections.

3.3.3 The Grid Checkpointer

The grid checkpointer is the service responsible for supervision of checkpoints for an application: it applies the checkpointing strategy to all running application units.

- It registers the application units with the checkpointer service on the nodes running the application's application units.
- It provides ressources to store the checkpoints.
- It detects node failure and takes appropriate mesures to restart the application. It must therefore manage the credentials of the user running the application to enable restart.
- It is able to launch applications in a checkpoint/restart context.
- It coordinates taking a checkpoint of an application running on different nodes.

This grid level checkpointer is the JobCheckpointing service described in WP3.3 The system level and kernel level checkpointers are described in more details in the next section.

3.4 Process Checkpointing in Linux-XOS

Process checkpointing in Linux-XOS will be based on BLCR, which is one of the most advanced open source implementation of a checkpoint/restart system for Linux. In particular, it is the only implementation with support for multi-threaded processes as well as for some implementations of MPI.

Our aim with T2.1.2 will be to augment BLCR with the following features:

- save the shared libraries used by the process in the checkpoint, rather than suppose that they will be present on the system when the process will be restarted.
- save the security context (VO specific information) in the snapshot of a process.
- at restart, provide information to the restarted process about the changes in the environment (process id, IP address, hostname).

As far as files are concerned, the truncate strategy will be implemented with more recent version of opened files overwritten upon restart.

3.4.1 The Kernel Checkpointer

The Kernel Checkpointer adds two functionalities to a standard Linux kernel:

1. bring a process to a checkpointable state,
2. save a snapshot of the state of a process to a file descriptor.

As bringing a process to a checkpointable state might involve some cooperation from the process, a mechanism to allow a process to register callbacks that should be called when checkpointing a process is added to the kernel, but this interface still needs some investigations.

These callbacks can be used by a process belonging to a grid application to implement any form of coordination that is needed to bring network connections or shared objects to a state from wich the grid application can be safely restarted.

Therefore, taking a checkpoint follows the following scenario:

1. A call to `checkpoint` a process is made.
2. The kernel calls all registered callbacks.
3. each callback does some work, then calls `checkpoint_ready`, which is a blocking call.
4. once all callbacks have finished preparing the checkpoint, application is stopped and a snapshot of the process is made.

5. callbacks return from `checkpoint_ready` restoring all information needed to restart or continue the process.

6. the process is restarted.

Therefore, the following API for the kernel checkpointer is envisioned:

- ```
  pid_t checkpoint (pid_t pid, int fd, int flags);
  ```
- ```
  checkpoint_ready() ;
  ```
- ```
  /* prevent checkpointing */
  int checkpoint_disable() ;
  ```
- ```
  /* enable checkpointing */
  int checkpoint_enable() ;
  ```
- ```
  /* attempt to restart checkpoint stored in fd */
  pid_t restart (fd, flags);
  ```
- ```
  /* attempt to restart process in checkpoint state
  (checkpoint_pid) with pid restart pid */
  pid_t restart (pid_t checkpoint_pid, pid_t restart_pid,
  int flags);
  ```

This API can be seen as a set of new system calls as well as the interface extending the standard library implemented in C (libc) that will be provided to application programmers.

Adding new system calls to Linux is a difficult task. Therefore, in order to ease acceptance of our work by the Linux community, our implementation is likely to add this API by other means than new system calls, such as the `ioctl` call often used by modules. Therefore, at first, implementers should rely on the C interface.

Details

- Any checkpointing requests that are made for an application while checkpoints are disabled are queued until checkpointing is enabled unless the NON_BLOCKING flag is provided, in which case taking the checkpoint will fail.
- On a call to `checkpoint`, if the KILL flag is set, the application is killed once the snapshot is taken.
- fd is an opened file descriptor.
- using the CLONE flags with `checkpoint`, the returned pid_t is the pid of a cloned process of pid, which can be checkpointed to a file descriptor later on, but not run, unless `restart` is called with that pid.

- The restart system call attempts to restart the checkpoint using the credentials of the process running the restart. The flags indicate if the restarted process should use the same pid (the call will fail if not possible) or if this doesn't matter.

3.4.2 System Checkpointer

The system checkpointer described here is viewed as an implementation of the `job_service` described in the Simple API for Grid Applications (SAGA [29]), that can handle a `job_description` which describes the way checkpointing should be implemented for the job. In that sense, we define the following API:

```
class checkpointable_job_description:
                    implements saga::job::job_description
{
  CONSTRUCTOR (out checkpointable_job_description obj) ;
  DESTRUCTOR  (in checkpointable_job_description obj) ;

  // Attributes:
  //   name: CheckpointPeriodicity
  //   desc: how frequently should the job be checkpointed,
  //         in seconds
  //   type: Int
  //   mode: ReadWrite, optional
  //   notes: - a value of 0 means no periodic checkpointing
  //          - default value is implementation dependant
  //
  //   name: NumberOfKeptCheckpoints
  //   desc: how many checkpoints should be kept for this job
  //   type: Int
  //   mode: ReadWrite, optional
  //   value: '1'
  //   notes:
  //
  //   name: FinalStorage
  //   desc: set of pathnames to use to store the checkpoint
  //   type: Vector string
  //   mode: ReadWrite, optional
  //   value: -
  //   notes: - if no path if given, a default path will be
  //            selected by the System Checkpointer,
  //            presumably on the local node
  //
  //   name: CheckpointPolicy
  //   desc: how the checkpoint is produced
  //   type: Vector string
  //   mode: ReadWrite, optionnal
  //   value: -
  //   notes: - if no policy is given, a default policy
```

```
  //            will be chosen
  //          - If more than one policy is given, the
  //            first policy available for the checkpoint
  //            service will be used
  //          - possible CheckpointPolicies include
  //             Safe: the checkpoint file is completly
  //               written before the checkpoint call
  //               returns
  //             LocalFirst: the checkpoint file is written
  //               locally before end of system checkpoint
  //               and moved to its final destination later
  //             MemoryFirst: the checkpoint is saved in
  //               memory at the end of the system
  //               checkpoint and moved to its final
  //               destination later
}
```

Using this API, a Grid Application will be able to use the System Checkpointer.

From the system's point of view, the System Checkpointer will be implemented as a service listening on a specific socket and using the SOAP protocol. Libraries implementing the described API will be available for different language bindings and will access the System Checkpointer in a transparent way.

3.4.3 Modifications to Linux

The basis kernel checkpointer (BLCR) was chosen such that it keeps modifications to the Linux kernel localized in one kernel module. This will avoid problems like trying to modify the core kernel and convince the Linux kernel developer community of the need to include this functionality. The kernel module can be maintained outside the kernel tree and built to fit particular kernel versions. The XtreemOS team will need to forward port the module to new kernel versions and/or collaborate with the BLCR developers from Berkeley Labs to keep the module in line with the kernel evolution.

Further changes are expected to be done in user space and collected in libraries linkable with applications or message passing libraries. These will again be managed inside the XtreemOS community and need no special acceptance by the Linux community. We aim to get the libraries and kernel module included into various Linux distributions.

3.4.4 Initial Thoughts on Advanced Functionalities

The main place of work for advanced functionnalities is around the checkpoint of files. Using support from the file system, it should be possible to checkpoint files in a more flexible way than the Truncate strategy described in this document.

The idea here is to have the file system save in the directory storing the checkpoint any data blocks belonging to the file that are modified after the checkpoint is

taken in a special file. Here, the overhead of the snapshot capability of the filesytem is supported in a user controlable way, because it is exposed as a file maintained by the file system but belonging to the user. Therefore, this file is managed by the user, and counts as a resource consumed by the user for any accounting system. Security implications of such a scheme need to be studied.

Chapter 4

Conclusion

In this document, we have specified Linux-XOS along two main directions: features to support virtual organizations and features to support application checkpointing.

Concerning the Linux support for virtual organizations, we have analyzed the requirements not only from XtreemOS use cases but also from other important research projects. From the state of the art on virtual organization management, it appears that though significant progress has been achieved with the development of flexible and scalable VO management frameworks and security services, there are several open issues left.

a) The establishment of a VO is still a heavy-weight and time-consuming process. Essentially, VO administration is not scalable with respect to the number of entities and resources involved.

b) It is also complex for Grid users to exploit VOs and manage VO identities.

c) There is still no perfect solution for node-level or site-level isolation of concurrent accesses from different VO users. Current VO middleware cannot fully leverage the native OS mechanisms to ensure security management and VO policy enforcement, at the node and site level, of a degree comparable with that of conventional operating systems.

We have distinguished two levels in VO management: VO level and node level. VO level management includes membership management of users and nodes that join in or leave from a VO, policy management(e.g. group and role assignment), and runtime information management (e.g querying active processes or jobs in a VO). VO-level management is done by XtreemOS-G services investigated in SP3 workpackages (WP3.2, WP3.3, WP3.4 and WP3.5).

The main responsibilities of node-level management include:

- mapping of, and transferring control from grid identities to local identities,
- applying VO policies that grant or deny access to local resources (files, services . . .),

- checking VO-mandated limitations of local resource usage (CPU wall time, disk quotas, memory ...),
- ensure protection and separation of resource usage by different Grid users,
- logging and auditing of resource usage.

Node level management requires support in the various flavor of XtreemOS-F. It is studied as part as WP2.1 and has been specified in this document.

In this document (see §2.3) we have give an overview of the basic implementation of VO support in XtreemOS, based on definitions of *Grid identity* and *Grid session* that are suitable to a scalable implementation.

Opening a new session on a Grid node takes the three classical steps: authentication, authorization and session creation. These three steps can be implemented in XtreemOS through extensions to the PAM plug-ins of Linux, or using a specific PAM-like service. The principal feature of the PAM approach is that the nature of the authentication is dynamically configurable. The system administrator is free to choose how individual service-providing applications will authenticate users using simple configuration files. The approach based on PAM modules is thus expandable and compatible with existing standards for identity certification on Grids.

In XtreemOS, a domain system administrator installs the plug-ins corresponding to the VOs having access to the local nodes. The system is evolutive: new plug-ins corresponding to new VO models (or credentials) can be developed and installed dynamically. Virtual organizations managed using different VO models can cohabit in the same node.

In order to run processes on a grid node, local entities like UID/GID couples must be allocated for each session, to map a Grid user identity to the local system. These UID/GID can be static (the user owns a local account on the node) or dynamic (the user is unknown). Ownership of a local account can be managed in a kind of gridmapfile (a configuration file listing user distinguished names and their local accounts) which is analyzed by a session PAM plug-in, while more complex plug-in can manage fully dynamic id mapping.

The correctness of the approach, when complex grid applications can be deployed across multiple grid nodes, heavily relies on the separation of the grid and local levels in resource management. As long as two grid sessions are opened on two different grid nodes, the only means of communication are through XtreemOS-G services. Access control to an object in XtreemOS depends on whether it is a kernel object or a grid object. Local files, processes, shared segments are all local objects on a grid node. In order to avoid deep modifications to the Linux kernel, access control to kernel objects should use the native DAC/ACL system of Linux in the XtreemOS basic version. Access control to grid objects will be managed inside XtreemOS-G services (grid file system, grid user certification authority and so on).

The existing mechanisms of security enhancement for Linux, such as Linux Security Module (LSM) framework [54] will be investigated to refine the access

control and resource enforcement mechanisms in advanced versions of XtreemOS (to be implemented in the second half of the project after M18). By extending the Linux operating system with built-in virtual organization support, XtreemOS can provide outstanding performance and enhanced security while minimizing administration costs of VO compared with existing middleware VO solutions.

Linux-XOS should implement methods and interfaces to checkpoint and restart applications. Application checkpointing in XtreemOS involves three levels of checkpointer: Kernel checkpointer, system checkpointer and grid checkpointer, the first two being implemented in XtreemOS-F, the last one being a service of XtreemOS-G (developed in WP3.3 as a part of the application management service).

Kernel-level checkpointing of processes in Linux-XOS will be based on BLCR, which is one of the most advanced open source implementation of a checkpoint system for Linux. In particular, it is the only implementation with support for multi-threaded application units as well a for some implementations of MPI. Our aim will be to augment BLCR with various features such as

- saving the shared libraries used by an application unit,
- saving the security context (VO specific information),
- extending the save functionality of the snapshot from a specific file (context.pid) to a generic file descriptor, so that checkpoints can be stored in a Grid Object in the future,
- at restart, providing information to the restarted application unit about the changes in the environment (process id, IP address, hostname).

The system checkpointer is viewed as an implementation of the `job_service` described in the Simple API for Grid Applications (SAGA [29]), that can handle a `job_description` which describes the way checkpointing should be implemented for the job (checkpoint frequency, storage policy for checkpoints, ...).

The basis of Linux-XOS Kernel checkpointer (BLCR) was chosen according to the global strategy of keeping changes to the Linux code base, as much as possible, minimal and local to a few kernel modules. This will avoid problems related to changes in the core kernel, which requires for each specific functionality to convince the Linux kernel developer community. The XtreemOS team will need to forward port the module to new kernel versions and/or collaborate with the BLCR developers from Berkeley Labs to keep the module in line with the kernel evolution.

The work performed on checkpointing application units in WP2.1 (on (individual PC running Linux-XOS) will be coordinated with the work done on checkpointing application unit on clusters in WP2.2 (clusters running LinuxSSI) even if we do not expect to have fully compatible mechanisms for both kinds of Grid nodes.

Major improvements of checkpointing functionalities on individual XtreemOS-F nodes will come from file-related issues. Exploiting support from the (Grid) filesystem, it should be possible to checkpoint files in a more flexible way than current systems do (e.g. BLCR's Truncate strategy). This is an advanced functionality that will be further investigated after M18.

By M18, we plan to be able to checkpoint/restart applications consisting of a single application unit (that is to say an application running entirely on a single Grid node). Checkpoint/restart of applications made up of multiple application units is considered as an advanced feature to be further investigated after M18.

Chapter 5

Glossary

AC Attribute Certificates

ACL Access Control List

AVC Access Vector Cache

BLCR Berkeley Lab Checkpoint/Restart

CA Certificate Authorities

CAS Community Authorization Service

CE Compute Element

DAC Discretional Access Control

DES Data Encryption Standard

DN Distinguished Name

EGEE Enabling Grids for E-sciencE

GSI Grid Security Infrastructure

HLA High Level Architecture

LCMAPS Local Credential Mapping Service

LDAP Light-weight Directory Access Protocol

LSM Linux Security Module

OGSA Open Grid Service Architecture

OGF Open Grid Forum

PDP Policy Decision Point

PIP Policy Information Point

PRIMA PRIvilege Management and Authorization

MAC Mandatory Access Control

NIS/YP Network Information System/Yellow Pages

PAM Pluggable Authentication Module

PERMIS PrivilEge and Role Management Infrastructure Standards validation

PMA Policy Management Authority

PKI Public Key Infrastructure

RB Resource Broker

RBAC Role Based Access Control

RPC Remote Procedure Call

SAGA Simple API for Grid Applications

SAML Security Assertion Markup Language

SE Storage Element

SSL Secure Sockets Layer

SSO Single Sign On

TLS Transport Layer Security

UML User-mode Linux

VOMS Virtual Organization Membership Service

XACML eXtensible Access Control Markup Language

Bibliography

[1] Akogrimo: Access to knowledge through the Grid in a mobile world. http://www.akogrimo.org/.

[2] EGEE: Enabling Grids for E-sciencE. http://eu-egee.org/.

[3] EGEE Project Technical Forum – Support. https://savannah.cern.ch/support/?group=egeeptf.

[4] Globus toolkit 4. http://www-unix.globus.org/toolkit/.

[5] Kerberos. http://web.mit.edu/Kerberos/.

[6] Liberty Alliance. http://www.projectliberty.org/.

[7] Linux-XOS Specification D3.5.1: State of the Art in Security for OS and Grids.

[8] Ogsa authorization wg. https://forge.gridforum.org/sf/projects/ogsa-authz.

[9] Openvz. http://openvz.org/.

[10] Security Requirements for a Grid-based OS.

[11] Virtual workspaces. http://workspace.globus.org/index.html.

[12] Vmware. http://www.vmware.com.

[13] Sundararaj A. and P. Dinda. Towards virtual networks for virtual machine grid computing. In *3rd USENIX Conference on Virtual Machine Technology*, 2004.

[14] R. Alfieria, R. Cecchinib, V. Ciaschinic, L. dellAgnellod, A. Frohnere, K. Loixrenteyf, and F. Spatarog. From gridmap-file to VOMS: managing authorization in a Grid environment. *Future Generation Computer Systems*, 2005(21):549–558, 2005.

[15] R. Butler, D. Engert, I. Foster, C. Kesselman, S. Tuecke, J. Volmer, and V. Welch. A National-Scale Authentication Infrastructure. *IEEE Computer*, 33(12):60–66, 2000.

[16] C. Le Goater, D. Lezcano, C. Calmels, D. Hansen, S. E. Hallyn, and H. Franke. Making Applications Mobile Under Linux. In *Proceedings of the Linux Symposium*, volume 1, pages 347–368, July 2006.

[17] Christopher D. Carothers and Boleslaw K. Szymanski. Checkpointing multithreaded programs. *j-DDJ*, 27(8):??–??, August 2002.

[18] Gerald Carter. *LDAP system administration*. O'Reilly & Associates, Inc, 2003.

[19] XtreemOS consortium. Annex 1 - description of work. Integrated Project, April 2006.

[20] W. R. Dieter and Jr. J.E. Lumpp. User-Level Checkpointing for LinuxThreads Programs. In *Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference*, pages 81–92, Berkeley, CA, USA, 2001. USENIX Association.

[21] J. Dike. User-mode linux. In *Proceedings of the 5th Annual Linux Showcase and Conference*, November 2001.

[22] Chadwick D.W. and Otenko O. The permis x.509 role based privilege management infrastructure. In *Proceedings of 7th ACM Symposium on Access Control Models and Technologies (SACMAT 2002)*, 2002.

[23] S. Farrell and R. Housley. RFC 3281:An Internet Attribute Certificate Profile for Authorization, 2002.

[24] I. Foster, C. Kesselman, and S Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of High Performance Computing Applications*, 15(3):200–222, 2001.

[25] I. Foster, H. Kishimoto, and A. Savva. The open grid services architecture, version 1.0. Informational document, Global Grid Forum, January 2005.

[26] M. Foster and J.N. Wilson. Pursuing the Three AP's to Checkpoint with UCLiK. In *Proceedings of the 10th International Linux System Technology Conference*, Saarbrucken, Germany, October 2003.

[27] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham, and T. S.Woodall. Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation. In *Proceedings, 11th European PVM/MPI Users Group Meeting*, pages 97–104, Budapest, Hungary, Septembre 2004.

[28] Roberto Gioiosa, Jose Carlos Sancho, Song Jiang, and Fabrizio Petrini. Transparent, incremental checkpointing at kernel level: a foundation for fault tolerance for parallel computers. pages 9–9, 2005.

[29] T. Goodale, S. Jha, T. Kielmann, A. Merzky, J. Shalf, and C. Smith. A Simple API for Grid Applications (SAGA). Grid Forum Working Draft GWD-R.72, Open Grid Forum, 2006. http://forge.ggf.org/sf/projects/saga-core-wg.

[30] H. Zhong and J. Nieh. CRAK: Linux Checkpoint/Restart As a Kernel Module. Technical Report CUCS-014-01, Department of Computer Science, Columbia University, November 2001.

[31] Ford W. Housley R., Polk W. and D. Solo. [RFC 3280] Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. 2002.

[32] Vollbrecht J., Calhoun P., Farrell S., Gommans L., and Gross G. RFC 2904:AAA Authorization Framework, 2000.

[33] 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.

[34] K. Keahey, I. Foster, T. Freeman, X. Zhang, and D. Galron. Virtual workspaces in the grid. In *Euro-Par 2005 Parallel Processing*, 2005.

[35] Pearlman L., V. Welch, I. Foster, C. Kesselman, and S. Tuecke. A Community Authorization Service for Group Collaboration. In *Proceedings of the IEEE 3rd International Workshop on Policies for Distributed Systems and Networks*, 2002.

[36] M. Lageman and S.C. Solutions. Solaris ContainersąłWhat They Are and How to Use Them. *Sun BluePrints OnLine*, pages 819–2679, 2005.

[37] M. Litzkow, T. Tannenbaum, J. Basney, and M. Livny. Checkpoint and Migration of UNIX Process in the Condor Distributed System. http://www.cs.wisc.edu/condor/doc/ckpt97.ps.

[38] Lorch M., Adams D. B., D. Kafura, Koneni M. S. R., Rathi A., and Shah S. The PRIMA System for Privilege Management, Authorization and Enforcement in Grid Environments . In *Proceedings of the Fourth International Workshop on Grid Computing (GRID03)*, 2003.

[39] OASIS. Xacml v1.0 oasis standard. http://www.oasisopen.org/committees/download.php/2406/oasis-xacml-1.0.pdf, February 2003.

[40] OASIS. Assertions and protocol for the oasis security assertion markup language (saml) v1.1. http://www.oasis-open.org/committees/download.php/6628/sstc-saml-tech-overview-1.1-draft-05.pdf, February 2004.

[41] S. Osman, D. Subhraveti, G. Su, and J. Nieh. The Design and Implementation of Zap: a System for Migrating Computing Environments. *SIGOPS Operating System Review*, 36(SI):361–376, 2002.

[42] Barham P., B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebar, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Proceedings of the nineteenth ACM Symposium on Operating Systems Principles (SOSP)*, 2003.

[43] M. rdos and S. Cantor. Shibboleth-Architecture DRAFT v0.5. http://shibboleth.internet2.edu/docs/draft-internet2-shibboleth-architecutre-05.pdf.

[44] Figueiredo R.J., P.A. Dinda, and Fortes J.A.B. A case for grid computing on virtual machines. In *Proceedings of 23rd International Conference on Distributed Computing Systems*, 2003.

[45] Sriram Sankaran, Jeffrey M. Squyres, Brian Barrett, Andrew Lumsdaine, Jason Duell, Paul Hargrove, and Eric Roman. The LAM/MPI checkpoint/restart framework: System-initiated checkpointing. *International Journal of High Performance Computing Applications*, 19(4):479–493, Winter 2005.

[46] S. Smalley, C. Vance, and W. Salamon. Implementing SELinux as a Linux Security Module. *NAI Labs Report# 01*, 43, 2001.

[47] Hal Stern, Mike Eisler, and Ricardo Labiaga. *Managing NFS and NIS*. O'Reilly & Associates, Inc, 2001.

[48] Barton T., Basney J., Freeman T., Scavo T., Siebenlist F., V. Welch, Ananthakrishnan R., Baker B., and Keahey K. Identity federation and attribute-based authorization through the globus toolkit, shibboleth, gridshib, and myproxy. In *Proceedings of 5th Annual PKI R&D Workshop*, 2006.

[49] M. Thompson, V. Welch, M. Lorch, R. Lepro, D. Chadwidk, and V. Ciaschini. Attributes used in ogsi authorization. www.ggf.org/documents/GFD.57.pdf, 2005.

[50] Mary Thompson, William Johnston, Srilekha Mudumbai, Gary Hoo, Keith Jackson, and Abdelilah Essiari. Certificate-based access control for widely distributed resources. In *Proceedings of the Eighth USENIX Security Symposium (Security '99)*, pages 215–228, 1999.

[51] Chadwick D. W., Novikov A., and Otenko O. Gridshib and permis integration. http://www.terena.nl/events/tnc2006/programme/presentations/, 2006.

[52] V. Welch, R. Ananthakrishnan, F. Siebenlist, D. Chadwick, S. Meder, and L. Pearlman. Use of saml for ogsi authorization. http://www.ggf.org/documents/GFD.66.pdf.

[53] V. Welch, F. Siebenlist, D. Chadwick, S. Meder, and L. Pearlman. Ogsi authorization requirements. http://www.ggf.org/documents/GFD.67.pdf.

[54] C. Wright, C. Cowan, J. Morris, S. Smalley, and G. Kroah-Hartman. Linux security modules: general security support for the linux kernel. *Foundations of Intrusion Tolerant Systems, 2003 [Organically Assured and Survivable Information Systems]*, pages 213–226, 2003.