Project no. IST-033576

XtreemOS

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

User's Guide

XtreemOS Technical Report # –NA–

Massimo Coppola

Report Registration Date: —

Version 0.01 / Last edited by Massimo Coppola / November 9, 2009

Project co-funded by the European Commission within the Sixth Framework Programme		
Dissemination Level		
PU	Public	√
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	28/04/09	Massimo Coppola		Initial document
0.15	13/05/09	Massimo Coppola		Second revision of the document; fixing issues and defined missing parts.
0.16	09/06/09	Ian Johnson	STFC	Changes to description of get-xos-cert - new arguments.

Contents

Chapter 1

Introduction

XtreemOS is a Linux-based operating system providing native support for virtual organizations (VOs) in next-generation grids. It represents a new approach to Grid and distributed computing, which overcomes many of the limitations of current middlewares. The XtreemOS Guides will let you make the first steps in using the system, and will lead you through all different installation phases, regardless that you are installing a full Grid platform, attempting an experiment with a few machines in your system, or just giving it a try on a Virtual Machine.

To better suit the different targets the Guides are split in a User's Guide, an Administrator's Guide, with several other related documents referenced from here.

1.1 What is in the Guides

The XtreemOS Administrator's and User's guides provide comprehensive guidelines for installing, configuring and operating an XtreemOS system.

The target of the **User Guide** are beginner users of the XtreemOS systems, who either:

1. already have access to an installed machine that is part of the XtreemOS network,
2. are trying a ready-made system-disk image on top of virtualization software such as VirtualBox, or
3. are using a live boot device that does not need installation (currently a CD-ROM, but USB flash device is planned).

User-level functionalities are explained and shown with examples, and common problems and basic configuration issues are either explained, or referenced in the Administration Guide.

The **Administration Guide** describes at length XtreemOS sytems installation procedures and system configuration, both using the graphical installer and discussing individual subsystem configuration files.

Both guides focus on the workstation and cluster (XtreemOS-SSI) flavours of XtreemOS, the distinction among them being almost completely confined to the Administration Guide. The mobile XtreemOS flavour is described in a separate document, which has as a prerequisite all the introductory part and the general concepts of the User's Guide.

1.1.1 Contents of the User Guide

- Chapter 2 describes what an XtreemOS grid looks like, its architecture and the core, resource and client services that must be deployed in order to operate the system
- Chapter 3 explains how to obtain the latest XtreemOS install CD and its software packages
- Chapter 4 describes how to operate an XtreemOS grid, its commands and the most common use cases
- Chapter 5 collects examples of practical use with small applications
- Chapter 6 collects frequently asked questions and common issues encountered during XtreemOS usage.

1.1.2 Contents of the Admininstration Guide

Chapters 2 and 3 are the same as the User Guide.

- Chapter 4.1 describes system installation from scratch using the graphical installer
- Chapter 5 explains how to run XtreemOS in just one machine, so you can get familiar with the main concepts and operations. TO BE REMOVED
- Chapter 6 explains how to set up a minimal XtreemOS grid in a quick and easy way
- Chapter 7 explains how to install and configure each XtreemOS service, in order to set up a full operative XtreemOS grid
- Chapter **??** collects frequently asked questions and common issues encountered during XtreemOS installation and administration.

Beside thes two guides, it is worth mentioning here the document which specifically targets the XtreemOS system flavour for mobile devices.

1.1.3 Note on conventions for showing command input

In the figures in this guide that show command input, commands which need root privileges to run are prefixed by a prompt 'Root# ' or '(root)# '. Long lines of command input may be entered on more than one line; the command is continued over to the next line by entering the backslash symbol so that the shell treats the input as one line. This is neccessary because of the constraints on column width in this Guide.

Consider the following command:

```
(root)# urpmi.addmedia xtreemos \
MIRROR/MandrivaLinux/devel/xtreemos/2009.0/i586/media/xtreemos/release/
```

This could entered to the shell by typing the backslash then a carriage return, and then entering the rest of the command. Alternatively, you could type the command as a single line.

1.2 Getting started

XtreemOS is a Linux-based operating system providing native support for virtual organizations (VOs) in next-generation grids. Unlike the traditional, middleware-based approaches, it is a prominent goal to provide seamless support for VOs, on all software layers involved, ranging from the operating system of a node, via the VO-global services, up to direct application support.

XtreemOS offers the following features and functionalities:

Application execution management

- Job basic manipulation : submit jobs, check job status, wait for job finalization, control jobs
- Send events to jobs
- Check, identify, reserve resources that match job requirements
- No global job scheduler is needed, the provided job management system is distributed

Virtual organization management

- A set of well-integrated security services
- Management of certificates
- Management of users, groups, roles, policies
- Management of resources

- Management of the complete lifecycle of VOs, operated via web and command-line interfaces[1]

Data management via the XtreemFS Grid file system

- Complete file system functionality across a Grid platform
- POSIX compliance
- Remote Grid authentication and authorization of users via XtreemOS certificates
- Volume creation and mount on any XtreemOS resource
- File striping and data replication, Grid file system monitoring

In the two System Guides you will see how these features are provided and how the necessary software is deployed within a Linux system. Of course, the most common ways of using XtreemOS will be connecting to an already existing XtreemOS platform, joining an XtreemOS platform with you machine, and building your own platform by installing a Linux distribution (like the Mandriva-XtreemOS) that already includes all the necessary XtreemOS packages by default.

1.2.1 Heterogeneous Devices

XtreemOS also integrates different *flavours* of the operating system, each one a different system distribution providing the XtreemOS functionalities on one of the different computing architectures that are commonly used in VOs.

- Stand-alone PCs (single CPU, or SMP, or multi-core) are supported by the standard XtreemOS distribution for laptops, which is the more heavily gorunded in the Linux operating system.
- For clusters of Linux machines, the XtreemOS-SSI flavour combines VO support with a single system image (SSI) functionality. A custom, SSI-enabled kernel is adopted to provide the functionality of a single Linux SMP machine as big as the supporting cluster, with a very large memory space, fully shared devices, and custom support for high-performance file system management as well as near-zero overhead transparent process migration.

  Apart from providing the abstraction of a single huge computing resource, the SSI platfrom is seamlessly integrated within the XtreemOS platform, and it is distributed together with the single machine version.
- Linux mobile devices, finally, can also easily join the XtreemOS platform. The XtreemOS-MD flavour provides them with VO support and specially-tailored, lightweight services for application execution, common data access, and user management.

[1] Note that managing the VO can be achieved via simple, scriptable command-line tools by the Administrator, when logged on the core node running the X-VOMS database. For maximum flexibility of use, the web interface can be made accessible from everywhere.

1.2.2 OGSA Compliance

In terms of the Open Grid Service Architecture (OGSA), as shown in the figure, XtreemOS is providing support on all layers involved in a virtual organization:

- On the *fabric layer*, XtreemOS provides VO-support by Linux kernel modules.
- On the *connectivity layer*, XtreemOS provides VO membership support for (compute and file) resources, application programs, and users.
- On the *resource layer*, XtreemOS provides application execution management.
- On the *collective layer*, XtreemOS provides the XtreemFS file system, and VO management services.
- On the *application layer*, finally, XtreemOS provides runtime support via the Simple API for Grid Applications (SAGA), next to native POSIX interfaces.

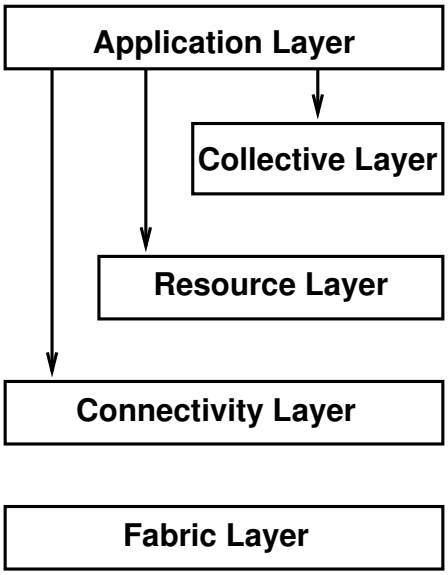

Figure 1.1: The layered Grid middleware architecture.

Chapter 2

Overview

In this chapter an overall description of the the XtreemOS system is given. We start from the three different kinds of nodes that compose the distributed platform, explaining their role and the additional system modules of XtreemOS that each kind needs with respect to a standard Linux system.

Section 2.2 discusses the different kind of cryptographic identity certificates that are exploited within XtreemOS to identify authorized users, and authenticate resources and services within the system.

Finally, section 2.3 will give a preliminary bird-eye description of the XtreemOS installation procedure.

2.1 Architecture

Within any XtreemOS system there are 3 different kinds of nodes (also called *configurations*):

- **Core** nodes, which are nodes providing the services that allow other nodes to provide resources to the XtreemOS system.
- **Resource** nodes, which are nodes providing resources to the XtreemOS system.
- **Client** nodes, which are nodes accessing the XtreemOS system without providing resources to it.

This is a logical functional division, whose purpose is to ease structuring the system and understandng its behaviour. Nothing precludes a certain machine from playing more than a single role, e.g. acting both as a client node and as a resource node, and this will be sometimes exactly the expected behaviour.

The important distinction in the platform is the one among core nodes and all other nodes. As it is for administrative accounts in a conventional operating system, it is generally advised to disallow arbitrary access on core nodes to users other than system administrators. Most services running on core nodes are critical for the functionality and security of the whole platform, thus security violations as well as core-node

downtime would have a negative impact on the XtreemOS system. This is nevertheless possible and allows to even demo-install XtreemOS on a single machine that provides all the services by itself.

Returning to the main topic, the different purpose of each configuration also determines which components and services should be present on it. In the following sections, we list and briefly describe these services for the Core, Resource and Client node configurations.

2.1.1 Core services

Core VOM services provide the certificate issuing and signing authorities for the users and the resources. They also provide the means to edit VO-level policies and making policy decisions. They are the following:

X-VOMS is the database containing all the information related to the virtual organizations living in the XtreemOS system

VOLifeCycle is a web frontend to manage the full lifecycle of virtual organizations, users and resources in the XtreemOS system. This includes a web interface for creating XOS-Certificates and private keys.

Credential Distribution Authority (CDA) server provides a way for the user to get an XOS-Certificate via the command-line CDA client.

Resource Certification Authority (RCA) server is used to get a resource certificate, in order to authenticate the resource in the XtreemOS system. RCA server comprises of the following: the RCA server logic, the front-end for both the RCA server logic and the RCA database implemented for DIXI, and the service certificate signed by the organisations root certification authority or another authority with the organisations root CA in the signature chain

Virtual Organization Policy Service (VOPS) serves requests and forwards answers from/to resource discovery services and digitally signs its decisions before forwarding responses back to services. VOPS enforces user requests against VO level policies for gaining access to specific resource nodes. VOPS is a standalone security service which provides Policy Administration Point (PAP), Policy Information Point (PIP), Policy Decision Point (PDP) to other XtreemOS services (e.g. AEM)

Core AEM services oversee the job submission process, select the nodes for the jobs and schedule their execution, and book-keep the jobs submitted so far.

Core XtreemFS services keep control of metadata as well as storage devices committed to the system and are the following:

Metadata Replica Catalog (MRC) stores the directory tree and file metadata such as file name, size or modification time. Moreover, the MRC authenticates users and authorizes access to files.

Directory service (DIR) is the central registry for all services in XtreemFS. The MRC uses it to discover storage servers.

2.1.2 Resource services

Resource AEM services or **ExecMng** receive and execute the scheduled jobs, monitor the resources and the job execution. Moreover, resource-level information services support distributed information management, setting up essential overlay networks within the platform and providing higly scalable management of resources. The latest category includes:

Service/Resource Discovery System provides the node services related to distributed information management

Resource Selection Service provides distributed, scalable and efficient resource location.

ADS provides an implementation of the Overlay Weaver and Scalaris DHTs that are compatible with the Application Directory Service (ADS) as provided by the SRDS package

Resource XtreemFS services or **Object Storage Devices (OSDs)** store arbitrary objects of files; clients read and write file data on OSDs.

2.1.3 Client services

Client VOM services are the following:

Credential Distribution Authority (CDA) client creates a private key and accesses the CDA server in order to get the user's XOS-Certificate, containing their public key and VO attributes.

ssh-xos allows login of Grid users into the system, with their XOS-certificate and access to their home XtreemFS volumes.

Client AEM services are the following:

XATI and C-XATI provide the service clients, letting the user perform administration tasks and exploit the virtual organization services.

xconsole_dixi provides a command-line interface for retrieving the available resources and submitting jobs.

XtreemFS client is implemented as a FUSE user-level driver that runs as a normal process. FUSE itself is a kernel-userland hybrid that connects the user-land driver to Linux' Virtual File System (VFS) layer where file system drivers usually live.

2.2 Certificates used in XtreemOS

XtreemOS uses X.509 v3 certificates to provide a Public Key Infrastructure for authentication. The certificates carry the public key of an entity and may be distributed to other parties that wish to establish the authentiticy of the key holder. The corresponding private key is secured by the key holder (the user) and is used to sign messages from them that can be verified by the public key. The conventional suffix for a certificate file is `.crt`, for a private key it is `.key`. Certificate signing requests (CSR files) have a `.csr` suffix.

All XtreemOS users need the XtreemOS root CA certificate installed on their machine. This is something that should be done by their system administrator.

End-users mostly deal with the XOS-Certificate. This identifies them to the system, and carries details of which VOs they belong to. In addition, if an end-user needs to use the VOLife web service via SSL (recommneneded), they can either import the XtreemOS root CA certificate into their browser, or accept the certificate presented by the VOLife web application the first time they connect to it using SSL.

System administrators of core nodes need to request service certificates from the XtreemOS root CA. Service certificates are currently used in the following ways;

- to authenticate servers to clients
- to authenticate clients to servers
- to sign a server response sent back to a client.

Most of the applications are services that run on core nodes. The XtreemFS mount client can run on any kind of node, and can authenticate to an XtreemFS server by sending the identity of the machine it is running on (using an service certificate), or it can send the user's XOS-Certificate to the XtreemFS server to prove the user's identity.

Resource administrators describe their resources by requesting resource certificates from the Resource Certificate Authority.

The fundamental step in setting up the XtreemOS certificates is for the root CA administrator to create the root certificate and corresponding private key, which is then used to sign requests for service certificates from system administrators. This root certificate has to be installed on every machine in the XtreemOS Grid, to enable checking of received service certificates and XOS-certificates.

- The XtreemOS root certificate needs to be installed by a system administrator on every machine in an XtreemOS Grid. It goes into `/etc/xos/truststore/certs/xtreemos.crt`.
- The default location for the user's XOS certificate is `$HOME/.xos/truststore/certs/user.crt`, with the corresponding private key in `$HOME/.xos/truststore/private/user.key`.

Table 2.1: XtreemOS Certificate types

Certificate Type	Used By	Purpose	Created-By, See Doc Sect.
Root certificate	CA admin, End-user	Public Key of the XtreemOS root CA. Can be imported into browser as a trusted certificate in order to access VOLife via SSL (https)	Root CA, §**ADM**:7.1
XOS certificate	End-user	Contains public key and VO attributes.	CDA server, §**USE**:4.1.1
Service certificate	System admin	Used to authenticate the identity of applications running on core nodes, or to authenticate an XtreemFS client).	Root CA, §**ADM**:7.3.5
Resource certificate	AEM	Attest to the resources provided by a resource node	RCA Server, §**ADM**:7.3.4

- Host certificates on a core node reside in `/etc/xos/truststore/certs/<service>.crt`. The corresponding private key is in `/etc/xos/truststore/private/<service>.key`. the `<service>` can actually be `cda`, `vops`, `rca`.
- The storage of resource certificates is handled transparently by the AEM RCA client.

2.3 Installation and configuration process

The installation and configuration process of a XtreemOS grid should follow the following order (please refer to section **ADM**:6 for minimal setup routine, or to section **ADM**:7 for a more detailed view of the process):

1. Install XtreemOS and configure the Root CA on on a single machine (see section **ADM**:7.1). This should not be a multiuser machine, and should be in a physically secure location. This machine should, ideally, not run any other XtreemOS services. To provide the ultimate in security (in order to avoid compromise of the Root CA), the machine need not be networked at all.
2. Install XtreemOS on a machine, and configure it as a core node to run the VO management services. Currently, the CDA and VOLife services have to be on the same machine as the X-VOMS database, unless remote database access has been configured. (Such configuration is outside the scope of this Guide.)

   Install the following VO Management services:

   - X-VOMS (section **ADM**:7.3.1).
   - VOLife (section **ADM**:7.3.3).

- CDA (section **ADM**:7.3.2).

This step requires using the Root CA to generate service certificates for the services (see section **ADM**:7.1.3).

Upon completing this step, you are in a position to start the X-VOMS, VOLife and CDA services. Users can register with the VOLife web application, create and join VOs, and define the groups they belong to, and roles they have, in VOs. Users can obtain XOS-Certificates (by using the VOLife webapp or CDA client) which contain their VO attributes and public key.

3. Install and configure node-level information services. These have to be present on the same machine(s) configured in 2. They build up overlay networks and provide services required by the AEM[1].

   - RSS (section **ADM**:7.5)
   - SRDS (section **ADM**:7.6).

   Upon completing this step you will be able to install node-level AEM services.

4. Install and configure AEM services. These could go on the same machine configured in 2 Above or a different machine.

   - RCA (section **ADM**:7.3.4).
   - VOPS (section **ADM**:7.3.6).

5. Install and configure XtreemFS servers (section **ADM**:7.7).

[1]Note: currently the SRDS and RSS services are implemented as node-level AEM services.

Chapter 3

Getting XtreemOS

This chapter is about getting XtreemOS binaries for installation, and how to proceed in the different situations (install from CD, repositories, disk images). You can find in chapter A a quick-reference of network URLs where to get binaries, sources, documentation and support.

3.1 The Install CD

The XtreemOS 2nd Public Release is based on the Mandriva Linux 2009.0 stable release. It includes all needed basesystem software needed to be able to run a basic GNU/Linux system with an X server, and all tools needed to be able to rebuild packages or software. The second CD includes all tools developped by the XtreemOS consortium, and all sources of those software as Source RPM packages.

The XtreemOS Install CDs are available from Mandriva mirrors. Several mirrors are available, you can find a list on this page : http://www.xtreemos.eu/software/mirror-websites

The ISOs are located in the MandrivaLinux/devel/iso/xtreemos directory.

Any further information for XtreemOS 2nd Public Release?

3.2 Disk images for virtual machine software

For the XtreemOS 2nd Public Release, disk images usable with common virtualisation software (e.g. Vmware, VirtualBox) will be made available for the sake of letting the user try a simple one-node XtreemOS installation without having to configure the system at all.

information needed

3.3 Main supported installation methods

3.3.1 Installation from CD

The most common method for installing XtreemOS GNU/Linux is using the first CD. Just burn the image you have downloaded from a Mandriva mirror. Starting an install from CD is as simple as putting the disc (the first disc) into the drive and rebooting.

3.3.2 Installation from hard disk

There are two major ways of doing a hard disk installation: you can either install from a local mirror of the XtreemOS GNU/Linux tree (which you have previously created by downloading the entire tree from a public mirror using, for e.g., rsync), or you can install directly from the .ISO format images of the XtreemOS GNU/Linux CDs without burning them to disc. To install from a local mirror, select the hard disk installation method. Then select the drive and partition where the local mirror is stored. Finally, enter the path to the correct directory of the XtreemOS GNU/Linux CD1. To install from .ISO images on a local hard disk, select the hard disk installation method. Then select the drive and partition where the ISO image is stored. Then enter the path to the ISO image.

3.3.3 Installation from a local network

Select the appropriate method for the type of server you wish to install from: NFS, HTTP, FTP. For all methods, you must now enter your network configuration information (for a typical home user, select DHCP and all default settings; other users should know their settings, or consult your network manager).

For the HTTP and FTP methods, you will now be asked to configure a proxy, if appropriate. If not, simply leave the boxes blank. For the HTTP and FTP methods, select the ”Specify the mirror manually” option. At the next step, enter the path to the mirror as appropriate. The path to enter is the path to the correct architecture (i586 only for the moment).

For the NFS method, after configuring networking, you will be asked to enter the hostname or IP address of the NFS server, and the path containing the XtreemOS GNU/Linux installation files.

3.3.4 From a Mandriva system

The packages of the XtreemOS 2nd Public Release are built on a Mandriva Linux 2009.0 distribution. It is therefore possible to install the packages on a Mandriva Linux 2009.0 distribution, see the next section about setting up packages repositories.

3.4 Setting up package repositories

Once your XtreemOS machine is installed, it is recommend to set up the online packages repository, to be able to install packages updates for bugfix.

This release of XtreemOS is based on Mandriva 2009.0, so you need to setup both Mandriva 2009.0 repositories, and XtreemOS repositories.

The following command will add both the Mandriva 2009.0 and XtreemOS repositories, selecting automatically the fastest mirror :

```
(root)# urpmi.addmedia --distrib --mirrorlist '$MIRRORLIST'
```

3.4.1 Using a proxy

If you are behind a proxy, you might need to setup wget or curl to use the proxy to let urpmi download the packages. This can be done by setting the following environment variables :

```
ftp_proxy=http://proxy_ip:port/
http_proxy=http://proxy_ip:port/
```

If using wget, this can be set in `/etc/wgetrc`, with the following variables :

```
ftp_proxy=http://proxy_ip:port/
http_proxy=http://proxy_ip:port/
```

Should not need to specify either option below, the default seems to work OK

Use the `--curl` or `--wget` option in your urpmi commands to select whether wget or curl should be used to download the packages.

3.4.2 The different types or repositories

The previous command will setup different kind of repositories :

- **release** The release repository contains the packages as they were when when the distribution was first released. This repository is not updated after the release.
- **updates** The updates repository contains the updated stable packages
- **testing** The testing repository contains testing package updates. Those packages should be used very carefuly. Normal users do not want this repository, for this reason it is not enabled by default.
- **backports** The backports repository is not available on the XtreemOS packages. It contains backports of new versions of some software. It should be used carefuly, for this reason it is not enabled by default.

Warning: Some testing and backport repositories are setup but not enabled by default. Make sure you don't enable them, or this could create some problems. This release of XtreemOS is based on Mandriva 2009.0, make sure you don't setup repositories from an other release such as 2009.1.

3.4.3 Updating packages

It is important to keep your system up-to-date. The following command will install the latest packages from the repositories you have configure:

```
(root)# urpmi --auto-update
```

3.4.4 Installing a package from the testing repository

If you want to install a package from the XtreemOS Testing repository, you can use the following command :

```
(root)# urpmi.update XtreemOS_Testing
(root)# urpmi --media XtreemOS_Testing package_name
```

3.4.5 Using URPMI

Documentation for urpmi is available in the following man pages :

- urpmi
- urpmi.addmedia
- urpmi.removemedia
- urpmi.update
- urpme
- urpmf
- urpmq
- urpmi.cfg
- urpmi.files

Chapter 4

Using XtreemOS

4.1 Virtual Organization Management

4.1.1 Obtaining user credentials in an XOS-Certificate

This section is for an 'ordinary' XtreemOS user who wishes to use a VO.

The first step in using XtreemOS is to register with your Grid, as described in section 4.1.4.

There are then two ways to obtain user credentials (private key and XOS-Certificate).

The Command-line Way The command-line CDA client program is called `get-xos-cert` and is invoked:

```
$ get-xos-cert host:port  voName
```

Figure 4.1: Obtaining an XOS-certificate from a CDA server

Where <host> should be replaced with the Fully-Qualified Host Name or IP address for the CDA server in your particular XtreemOS Grid.

The parameter <port> should be replaced with the numeric port that the CDA server is listening on. Your Grid administrator can provide these details.

The parameter <voName> should be replaced with the name of the VO that you wish to consider the primary VO.

The `get-xos-cert` command can also take additional arguments.

- **-k** Specify location of private key output file
- **-c** Specify location of certificate output file
- **-K** Specify location of an existing private key to use as input

- **-g** Specify a group name to consider as the primary group for this certificate
- **-t** Test mode: specify username and password for pre-configured user

The command prompts for the username and password that the user has registered with the VOLife webapp. If the correct details are provided, the command then generates a private key for the user, stored in the users home directory `$HOME/.xos/truststore/private/user.key`. You will be prompted for a passphrase to protect the private key. An XOS-Certificate is then obtained from the CDA server and stored in `$HOME/.xos/truststore/certs/user.crt`.

The XOS-Certificate is validated and verified by the `get-xos-cert` program before it is stored into the `user.crt` file. If you wish to examine the XOS-Certificate later, the command to use is:

```
view-xos-cert $HOME/.xos/truststore/certs/user.crt
```

Figure 4.2: Viewing an XOS-Certificate

This command prints a header 'XtreemOS Attributes' before the certificate extensions which store the VO attributes.

The Web Way The user credentials can be generated via the VOLife webapp in two steps, as described in section 4.1.4, by following the operations 'Generate new keypair' and 'Generate an XOS-Cert'.

4.1.2 Operating the Root Certificate Authority

This section is for the grid administrator, that is, the system administrator of the machine running the XtreemOS Root Certificate Authority.

The main operating mode of the XtreemOS Root CA is to take Certificate Signing Requests (CSR files) for applications and convert them to service certificates. The CSR file can be sent to the administrator of the root CA in an email message or by other means. The operator of the root CA should save the CSR file and examine it to check its validity, and contact the originator of the request if necessary (to verify its source).

For example, figure 4.3 shows the contents of a request for the CDA server on the host host.org.domain:

```
$ view-csr host.org.domain-cda.csr
Certificate Request:
    Data:
        Version: 0 (0x0)
        Subject: CN=host.org.domain/cda, O=XtreemOS Project,OU=cda
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:d0:ec:ed:89:93:2e:c3:23:47:7c:30:1e:de:fb:
                    40:bb:9a:6f:bd:77:35:54:28:24:b2:62:9b:cc:9e:
                    dd:f5:1b:19:55:be:fe:0b:9f:2d:56:a6:98:bd:77:
                    53:1c:da:38:3a:ba:60:03:90:f9:bc:a4:af:ec:5a:
                    c1:ec:80:34:cb:bd:fa:18:93:af:c1:84:5d:16:72:
                    ed:94:e8:eb:59:13:1f:99:6b:ac:93:d3:e5:07:e1:
                    54:77:e8:8f:44:4c:4a:0b:31:5c:26:af:19:c3:f6:
                    6c:71:22:cb:0c:56:47:99:f3:14:ab:1b:43:de:e9:
                    13:48:17:00:f0:da:0c:de:e1
                Exponent: 65537 (0x10001)
        Attributes:
        Requested Extensions:
            X509v3 Basic Constraints:
                CA:TRUE
            X509v3 Subject Alternative Name:
                DNS:host.org.domain
    Signature Algorithm: sha1WithRSAEncryption
        5a:c2:4e:aa:8f:bc:e2:c5:b2:0a:12:20:92:d5:90:de:fb:96:
        bb:d7:c3:5f:6d:67:89:5e:b1:6c:e1:9e:c6:e7:8e:e9:42:ea:
        0e:65:f1:3d:e9:73:31:44:95:6c:d3:3c:b4:b1:cc:bb:e6:1c:
        3c:a2:c1:99:a7:2e:d4:24:10:06:49:99:51:94:53:81:d9:8e:
        4d:a2:f5:1c:ac:df:19:e0:f4:bb:96:19:5f:74:88:57:e3:82:
        01:e7:93:a1:45:cc:3e:ef:54:28:bb:8a:1e:c0:3a:a9:dd:85:
        00:2f:ac:c7:b8:5c:c8:94:99:2f:a7:04:76:d4:74:84:f4:5d:
        a7:92
$
```

Figure 4.3: Examining an service Certificate Signing Request.

Note that the Subject Alternative Name extension field contains the Fully-Qualified Host Name of the machine that this certificate will be used on. This hostname, along with the name of the application, is also encoded in the Subject CN field.

If you trust the originator of this request, and you have validated the request by examining it as shown above, you are now in a position to generate an service certificate from the request. Figure 4.4 shows the step needed to create an service certificate from

a CSR file, in this case for the CDA server at host.org.domain.

```
process-csr /opt/xtreemosca host.org.domain-cda.csr
```

Figure 4.4: Signing a host CSR file, creating a service certificate.

The service certificate is created in the file `host.org.domain-cda.crt` in the current working directory. A copy of the service certificate is also stored in the 'certs' subdirectory of the root CA. In the example above, this would be `/opt/xtreemosca/certs`), named `<NN>.pem`. A record of the newly issued service certificate is made in `/opt/xtreemosca/index.txt`, including the certificate's expiry date, revocation status, serial number and CN.

4.1.3 Using X-VOMS

Utilities

`/usr/share/xvoms/bin/xtreemfsclient.sh` a small utility for testing XtreemFS volume creation via X-VOMS. This utility assumes the configuration files (listed below) locate at /usr/share/xvoms.

- hibernate.cfg.xml
- log4j.properties
- MRC.properties

Here is how to use this utility:

```
Usage: ./xtreemfsclient.sh -[nu] arguments
-n <URL of mrc> <volume name>:   create a named volume.
-u <username>:   create a new test user and his volume.
(password is set to be the same as the username.)
```

Option -u reads MRC.properties which should contain the following information:

```
mrc.host=localhost
mrc.port=32636
```

Note: You should change this file to fit your local setting.

4.1.4 Using VOLife

Using the web frontend of VOLife

here a section is missing, isn't it?

Quick Start

- If both MySQL and Tomcat services are running, open web browser to access the URL where the VOLife service for your Grid is running:

  `http://volifehost:8080/volifecycle/`

- By default, Tomcat listens on port 8080 for accepting HTTP requests. If your Grid administrator has set up Tomcat with HTTPS, the port number may be different.

- The first step of starting VOLife is to create an account by click "`create an account`" menu item on the left panel. After you create the account successfully, use it to sign in the website.

- The VOLife page offers a link to obtain a copy of the Root certificate for your Grid. You can save the certificate to your local machine by right-clicking on it, and then copy it (as user 'root') to **/etc/xos/truststore/certs/xtreemos.crt**.

Operations

There are three categories of functionalities provided by VOLife. The current implementation of VOLife supports the following operations:

Home → About me By clicking this menu item, a user can find out his GUID (Global User Identifier) and his home volume URL. Such information is useful when a user wants to manually mount his own XtreemFS volume.

Home → Create_a_VO After logged in with VOLife, a user is able to create a VO of her own by providing the VO name and its description. The user naturally become the owner & member of this vo, i.e. she is able to get her XOS-Cert.

Home → Join_a_VO A user could choose one VO (created by others) from the list of VOs in current XVOMS database, then click JoinVO button to submit joining request. Note that LeaveVO button is not implemented yet. The joining request is to be approved by the owner of that VO.

Home → My_Pending_Requests List all pending requests of current user. Requests could beVO joining or resource joing request.

Home → Get_an_XOS-Cert The one-step operation to get an XOS-Cert. Choose a VO to which the current user belongs and specify the passphrase for current user's private key. If the private key does not exists, it will be automatically created. If the generation is successful, there will display a link to download the generated cert file.

Home → Generate_new_keypair This is an optional functionality to generate user's private key and download it. This is not as secure as using the `get-xos-cert` command, as the private key is downloaded from the network.

Manage_My_VOs → My_Owned_VOs List all VOs owned by current user. Note that DeleteVO function is not implemented yet.

Manage_My_VOs → Approve_Requests List all requests submitted by other users and approve them.

Manage_My_VOs → Manage_Groups_Roles Manage groups and roles in VOs owned by current users. There could be multiple groups in one VO and there could be multiple roles in one group. Each group or role is actually a set of users. To add users into groups or roles, click VO,Group or Role names to list users belong to them, and right click add their subitems or users.

Manage_My_VOs → Manage_Policies Not implemented yet. This is to be integrated with VOPS service.

Manage_My_Resources → Register_a_RCA Add a RCA item in XVOMS database. Note that this RCA information is not actullay linked to (or relies on) RCA server. It is just an information item kept in database for future use.

Manage_My_Resources → Add_a_resource Add a resource into a RCA. A resource is a node with specific information like ip address and capability. To make a resource join to a VO, click AddtoVO button to submit the request.

Manage_My_Resources → Approve_resources Approve resource joining requests.

Manage_My_Resources → Get_Machine_Certificates Not implemented yet. It is to be integrated with RCA service.

Known Issues

is the following still true in XtreemOS 2nd Public Release?

Due to the time constrain, some of the security features have not been able to be incorporated into the *current* VOLife release. Specifically, there are two limitations:

- user information (including password) are transmitted over the network in plain text;
- only users are authenticated (via username/password challenge) but there is no way for users to authenticate the server running VOLife. That is, users might be tricked into divulging their username/password to VOLife on a hijacked server.

Both of these problems can be remedied by the introduction of https protocol to VOLife, which we plan to support in the next release.

Using the backend of VOLife

The backend of VOLife is wrapped into a command line uitility, to launch this:

```
volife_run.sh [options]
```

With no arguments provided, a list of available options is printed as follows on the screen, which has almost the same functionalities as the web frontend (see the following option listings) .

```
Usage: java XVOMSUtil [options]
[options]
        -create-user <user_name> <password> <first_name> <last_name> <organization> <email>
                Register a new user
        -list-all-users
                List all registered users in current database
        -list-user <user_name|guid>
                List a user with given user name or guid
        -approve-user <user_name|guid>
                Validate a registered user
        -create-vo <vo_name> <description> <vo_owner_name|vo_owner_id>
                Create a vo with specified owner
        -list-all-vos
                List all VOs in current database
        -list-vo <vo_name|vo_id>
                List a VO with given vo name or gvid
        -list-vo-of-user [owner_name|owner_guid]
                List all VOs owned by a given user
        -list-user-of-vo [vo_name|vo_gvid]
                List all users in a given VO
        -list-all-reqs
                List all requests in current database
        -create-user-req <vo_name|vo_gvid> <user_name|user_guid>
                Create a user joining request
        -list-user-req <vo_name|vo_gvid>
                List user joining requests of a VO
        -approve-user-req <requst_id>
                Approve a user joining request by an id
```

```
-create-rca <name> <description> <host> <port> <rca_owner_name|rca_owner_guid>
        Register a RCA site with specified owner
-list-all-rcas
        List all registed RCAs in current database
-list-rca <rca_id>
        List a RCA with given id
-list-rca-of-user <user_name|user_guid>
        List all RCAs owned by a given user
-create-resource <name,description,host,port,os,arch,speed,cpus,ram> rca_id>
        Add a new resource into a RCA
-list-all-resources
        List all resources in current database
-list-resource <rca_id>
        List all resources of a RCA
-create-resource-req <vo_name|vo_gvid> <resource_id>
        Create a resource joining request
-list-resource-req <vo_name>
        List all resource requests of a VO
-approve-resource-req <requst_id>
        Approve a resource joining request
-add-user <vo_name|vo_gvid> <user_name>
        Directly add a user into a vo (no need to approve)
-add-group <vo_name|vo_gvid> <group_name>
        Add a group into a VO
-list-group <vo_name|vo_gvid>
        List groups of a VO
-add-role <vo_name|vo_gvid> <group_name> <role_name>
        Add a role into a group of a VO
```

```
-list-role <vo_name|vo_gvid> <group_name>
        List roles of a group of a VO
-assign-group <user_name|user_guid> <vo_name:group_name>
        Assign a user with a group
-assign-role  <user_name|user_guid> <vo_name:group_name:role_name>
        Assign a user with a role
-convert-user <user_name|user_guid> <vo_name|vo_gvid>
        Convert from User in XVOMS to VOUser in CDA
-gen-keypair <user_name|user_guid> <password>
        Generate private keypair for a given user
-gen-xoscert <user_name|user_guid> <vo_name|vo_gvid> <passphrase> [valid_days]
        Generate XOS-Cert for a given user in a VO
```

List of options of XVOMSUtil

4.1.5 RCA

Resource Certification Authority (RCA) is a security service being developed in WP3.5. The purpose of this service is to provide a base to bootstrap the trust of resource nodes in XtreemOS. Organizations can provide computational resources to allow users to exploit the computational capabilities offered by their machines. In XtreemOS, this is facilitated by the Application Execution Management (AEM) services.

The RCA services consist of the **RCA server** which includes the **RCA database**, and of **RCA client**.

RCA server with RCA DB.

The **RCA server** is the core-level service which usually runs on a single well-secured node within a physical organisation. The service is interacted by X-VOMS with notification on VO membership changes on the machine, and by RCA client with requests for certificate signing. The service does not interact directly with the users, but rather depends on other services for the interaction. However, with the **dixi-xati** package, there are several scripts that can be used for testing RCA server:

- **rca_confirm** *Resource_Address* — Confirm the registration of the node in RCA DB. The *Resource_Address* needs to be of the form *IP*:*Port*, and the values of *IP*:*Port* can be found on the **rca_list_pending** output list. The command follows an RCA client's application for node registration action.
- **rca_list_pending** — Show the list of applied nodes in RCA DB.
- **rca_list_registered** — Show the list of registered nodes in RCA DB.

The RCA server can hold a list of VOs that the resources can join. The **rca_vo** command can manipulate this:

- **rca_vo a** *VO_id* — include the VO with the *VO_id* on the list of the VOs available to the resources in the RCA.
- **rca_vo r** *VO_id* — remove the VO with the *VO_id* from the list of the VOs available to the resources in the RCA.
- **rca_vo l** — retrieve the list of the VOs available to the resources in the RCA.

The RCA DB holds the collection of resources, their details as provided by the client at the registration or by VOLife, and each resource in the collection has a status which can be either *pending for confirmation* or *registered*. A resource that has been registered can have its certificates signed by the RCA server. The RCA DB temporarily also holds the information on the VOs the resources are members of. The membership of the machine in the RCA is controlled externally, via VOLife. However, there are the following utility commands that can simulate the involvment of the current node in the VO lifecycle:

- **rca_resource_vo a** *VO_id* **[***Resource_Address***]** — set the node denoted with *Resource_Address* in the form of *IP*:*Port* to be a member of the VO with *VO_id* for its id. If the *Resource_Address* parameter is omitted, the current node will be added. If the target node is online, the result of the script invocation will be a new VO-related attribute certificate placed into the node's folder defined in **RCAClientConfig.conf** as **resVOAttributeCertIncoming**. To install it, simply move it to the same folder that contains the machine identity certificate.
- **rca_resource_vo r** *VO_id* — remove the current node from a membership in the VO with *VO_id* for its id.
- **rca_resource_vo c** *VO_id* — request a VO-related attribute certificate of VO with *VO_id* for its id for the current node. The result of the script invocation will be an installed new VO-related attribute certificate.

The test VO has the id `0d17b96e-89d8-4e1b-91ec-10a6a93e9996`.

RCA client.

The RCA client helps manage the machine certificates (i.e., machine's identity certificate, its attribute certificate and all the VO-related attribute certificates). It generates the private key for the machine certificates and has it signed by the RCA server. It also collects the information on the current node from AEM's Resource Monitor to make the process of the resource registration simpler, and provides the retrieval of the local machine identity and VO-related attribute certificates to VOPS.

For the machine management enabled by the RCA, the following XATI scripts can be used:

- **dixi_test -RCA i** — The RCA client holds the information on the current node. This command lets the user choose the manner of the obtaining of the node's information. Primarily, the information can arrive from AEM's Resevation Manager. This information can be stored into /etc/xos/config/ResourceDescriptor.xml and modified in a text editor, then used in the future in a modified form.
- **rca_apply** — Apply for registration of this node with RCA DB.
- **rca_request** — Request a new resource certificate pair. The script has the RCA client create a new private key, sends the corresponding public key for signing to RCA server and, if properly registered in RCA DB, lists the obtained certificate contents. It creates and replaces the files containing the machine certificate and machine's private key into the files configured in **RCAClientConfig.conf** as **resIdentityCertFileName** and **resPrivateKeyFileName**, respectively. It also stores the machine's attribute certificate into file defined as **resAttributeCertExtFileName** (unless the RCA server returns a V2 attribute certificate; in this case the file is saved into the path and filename defined by **resAttributeCertFileName**).
- **rca_info** — Display the current resource certificate pair details.

Interactions between resource admin and RCA.

The resource administrator is the person in charge of a node (a machine) that is capable of providing services for job execution in the XtreemOS. Before the node can take part in a VO, it needs to be registered with the RCA and obtain the machine certificates. Figure 4.5 shows a diagram with the interaction between administrators and the components of the RCA.

The machine runs an RCA Client which helps the resource administrator to interact with the RCA Server. The RCA Server is a core-level service which runs on one or few select nodes within the local organisation's network. The RCA Client collects the data on the node and, by resource administator's request, applies for registration with RCA DB. Once the orgaisation-level administrator confirms the registration, the resource administrator can have the RCA Client create the machine's private and public key pair, and request a signed machine identity certificate and the attribute certificate from the RCA server.

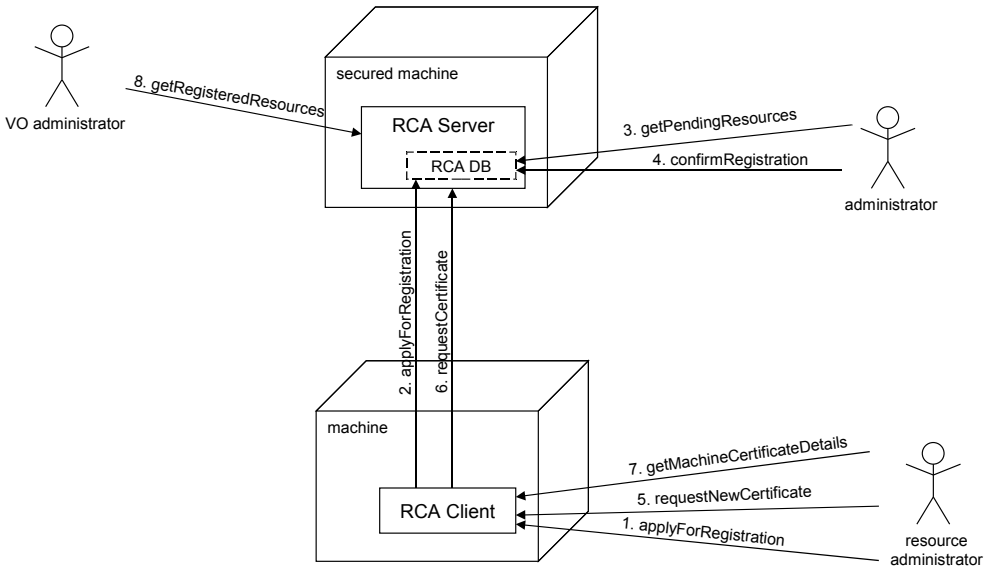

Figure 4.5: Interactions between resource admin and RCA server

4.1.6 Using VOPS through XConsole

Primary intention of having an entity acting as a policy service is to support coordinated access control over VO resources by offering VO level policy decision point

(PDP). Such access control can be performed on individual resources and can be used as access control to a group of resources/services sharing certain characteristics. VO-level policies and node-level policies form a hierarchical access control framework that can be tuned to achieve various degrees of control to resource usage within a VO.

Three major components comprise VOPS service: Policy Information Point (PIP), Policy Decision Point (PDP) and Policy Administration Point (PAP).

Currently PIP uses certificates provided by other services to extract all necessary information for performing control access over resources. In the future, all missing attributes not provided with the request as additional parameters, will be fetched through ADS or some other mechanism.

PAP is a set of methods which are used by administrator to control/manipulate XACML database. If user has appropriate rights (is a VO or resource administrator), she can use one of these commands using XATI's **xconsole_dixi** package:

- **xlistPolicy -pid** $\langle policyId \rangle$ lists policy with specified **policyId**, which is stored in the VOPS policy storage. Policy is listed in XACML format.
- **xlistPolicies** lists all policies stored in the VOPS policy storage. Policies are listed in XACML format.
- **xaddRuleToPolicy -rule** $\langle xacmlRule \rangle$ **-pid** $\langle policyId \rangle$ adds rule which is passed as XML string in XACML format (**xacmlRule** is a path to XACML file containing XACML rule) to the policy residing in VOPS policy storage and identified by **policyId** argument. Returns rule created as a string object. See also example on picture 4.7.
- **xremoveRuleFromPolicy -rid** $\langle ruleId \rangle$ **-pid** $\langle policyId \rangle$ removes rule identified by **ruleId** from policy with **policyId**.
- **xremovePolicy -pid** $\langle policyId \rangle$ Removes policy identified by **policyId** from policy storage
- **xaddPolicy -policy** $\langle xacmlPolicy \rangle$ adds policy residing localy on the path **xacmlPolicy** into VOPS policy storage. See also example on picture 4.6.
- **xreloadVOPS** reloads VOPS policy storage.
- **xwritebackVOPS** writesback VOPS policy storage to directory where **policyStorage** entry in VOPS config file points to.

PDP consists of methods of the VOPS framework which are used by invoking services requesting control access over a query for e.g. user accessing some resource or user trying to submit a job on a particular resource. These PDP methods of the VOPS are used internally by AEM through Resource Manager service.

By default VOPS provides a default policy wich permits any action of a user to any resource. This default policy is listed in figure 4.6. Other policies can be added using PAP and commands listed above. More refined rule, which can be injected into policy, is listed in figure 4.7.

```
<Policy PolicyId="Policy01" RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides">
  <Description>
        This is the default policy - it permits access for all users to all
        resources.
  </Description>
  <Target>
    <Subjects>
      <AnySubject/>
    </Subjects>
    <Resources>
      <AnyResource/>
    </Resources>
    <Actions>
      <AnyAction/>
    </Actions>
  </Target>
<Rule RuleId="DefaultRule" Effect="Permit">
    <Description>This rule permits access.</Description>
    <Target>
      <Subjects>
        <AnySubject/>
      </Subjects>
      <Resources>
        <AnyResource/>
      </Resources>
      <Actions>
        <AnyAction/>
      </Actions>
    </Target>
  </Rule>
</Policy>
```

Figure 4.6: A default XACML policy. This policy permits all actions to all users on a resurce in query.

4.2 Checkpointing

4.2.1 General requirements

Job checkpoint meta-data will be stored under */xtreemfs/job_checkpoint_meta_data*. Assure, this directory exists. Take care, cgroup support has been deactivated. (/etc/xos/nss_pam/pam_xos.c 'UseCGroups no')

4.2.2 BLCR

Two BLCR kernel modules (*blcr* and *blcr_imports*) must be loaded for BLCR to work. Assure, they are loaded:

```
$> lsmod |  grep blcr
```

If they are not shwon, identify the blcr source directory ...

```
$> find / -name cr_module
```

... and load the modules from there.

```
$> make insmod
```

BLCR stores its checkpoint images under **/xtreemfs/blcr**. Assure, this directory exists. Applications to be checkpointed with BLCR must be statically linked against the BLCR library.

```
$> gcc -o aout aout.c -lcr
```

Assure, the executable listed in your jsdl file fulfills this requirement, otherwise restart will fail.
To test BLCR cp/rst you can use the BLCR test suite. It is located under BLCR_SRC/examples.

```
  <Rule RuleId="SampleRule01" Effect="Deny">
    <Description>
     This rule denies all actions on resources inside VO:3ffebd8b-d110-4cf8-aae3-2d9a8a61564c
to a user with globalUserID = 2680f7fb-c6fd-479b-8579-03c782f63545.
    </Description>
    <Target>
  <Subjects>
      <Subject>
        <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
    3ffebd8b-d110-4cf8-aae3-2d9a8a61564c</AttributeValue>
          <SubjectAttributeDesignator AttributeId="subject:extensions:globalPrimaryVONname"
    DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </SubjectMatch>
        <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">2680f7fb-c6fd-479b-8579-03c782f63545</AttributeValue>
          <SubjectAttributeDesignator AttributeId="subject:dn:id-at-commonName"
    DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </SubjectMatch>
        <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">2680f7fb-c6fd-479b-8579-03c782f63545</AttributeValue>
          <SubjectAttributeDesignator AttributeId="subject:extensions:globalUserID"
    DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </SubjectMatch>
      </Subject>
    </Subjects>
      <Resources>
        <Resource>
          <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">3ffebd8b-d110-4cf8-aae3-2d9a8a61564c</AttributeValue>
            <ResourceAttributeDesignator AttributeId="resource:extensions:VO" DataType="http://www.w3.org/2001/XMLSchema#string"/>
          </ResourceMatch>
        </Resource>
      </Resources>
      <Actions>
        <AnyAction/>
      </Actions>
    </Target>
    <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">WINNT</AttributeValue>
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
          <ResourceAttributeDesignator AttributeId=
    "resource:jsdl:JobDefinition:JobDescription:Resources:OperatingSystem:OperatingSystemType:OperatingSystemName"
     DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </Apply>
      </Apply>
    </Condition>
  </Rule>
```

Figure 4.7: A sample XACML rule which denies any action over a resource from distinct VO. Request must contain attribute OperatingSystemName which equals WINNT (e.g. extracted from JSDL when submitting a job).

The following applications are available: counting, pthread_counting (multithreaded cp/rst), file_counting (file cp/rst), pipe_counting (multi-process cp/rst).

4.2.3 LinuxSSI

The LinuxSSI checkpointer stores checkpoint images unter */var/chkpt*. Assure, this directory exists.
To test LinuxSSi cp/rst you can use the LinuxSSI test suite. It is located under svn gforge.inira.fr xtreemos/grid/testing/checkpointing.

4.2.4 Using the grid checkpointer

The grid checkpointer can be called from within the dixi Xconsole via:

```
$> xcheckpoint jobID
$> xrestart jobID checkpointVersion
```

For checkpointing/restarting a distributed application, directories

- /xtreemfs/job_checkpoint_meta_data
- /xtreemfs/blcr
- /var/chkpt

must be mounted as XtreemFS volumes.

4.3 Application Execution Management

The interaction with the AEM occurs through the use of XATI as programming interface. From the user's perspective there is also a set of command line applications to ease the typical job submission, control and monitoring.

A key element of the interaction with AEM is the job description language (JSDL) which is described next. Afterwards, we describe the command line utilities of AEM, the xconsole_dixi, and some advanced features that allow the user more refined control of the system.

4.3.1 Job description

The job submission needs a job description that contains the details on the processes being run as well as the resource requirements needed by the job. AEM uses JSDL to store the description and the requirements. Figure 4.8 shows an example of a JSDL document that describes a job with its executable, defines its standard output and standard error stream files and in this case has no specific resource requirements.

Arguments and environment variables

Other useful entries in the POSIXApplication part of the JSDL are Argument and Environment tags. Note that a new Argument tag is required for every parameter of the application, putting more than one in the same tag is like using quotes in a shell to allow parameters with spaces inside.

The format of the Environment tag requires a name attribute to specify the environment variable name. See the example in Figure 4.9 which would be equivalent to running this on a shell:

```
GREP_COLORS="mt=01;31:ml=:cx=:fn=35:ln=32:bn=32:se=36" \
grep --color "Hello world" test-file
```

```
<?xml version="1.0" encoding="UTF-8"?>
 <JobDefinition xmlns:jsdl="http://schemas.ggf.org/jsdl/2005/11/jsdl">
   <JobDescription>
     <JobIdentification>
       <Description>Show me the calendar</Description>
       <JobProject>Calendar</JobProject>
     </JobIdentification>
     <Application>
       <POSIXApplication
           xmlns:ns1="http://schemas.ggf.org/jsdl/2005/11/jsdl-posix">
         <Executable>/usr/bin/cal</Executable>
         <Output>out_ls.txt</Output>
         <Error>err_ls.txt</Error>
       </POSIXApplication>
     </Application>
   </JobDescription>
 </JobDefinition>
```

Figure 4.8: A typical JSDL for submitting a job with no resource requirements.

```
<jsdl-posix:POSIXApplication>
    <jsdl-posix:Executable>grep</jsdl-posix:Executable>
    <jsdl-posix:Argument>--color</jsdl-posix:Argument>
    <jsdl-posix:Argument>Hello world</jsdl-posix:Argument>
    <jsdl-posix:Argument>test-file</jsdl-posix:Argument>
    <jsdl-posix:Environment name="GREP_COLORS">
        mt=01;31:ml=:cx=:fn=35:ln=32:bn=32:se=36
    </jsdl-posix:Environment>
    <jsdl-posix:Output>test-output</jsdl-posix:Output>
</jsdl-posix:POSIXApplication>
```

Figure 4.9: A POSIX Application definition to be used inside a JSDL.

Table 4.1: Command line utilities for AEM

Program name	Description
xsub	Submits a job to the grid
xsub.sh	Generates a JSDL for an executable file and submits it to the grid
xps	Gives information about the jobs in the grid, monitoring client
xjobKill	Sends events to a submitted job
xwait	Waits for a job's end and gets its exit status
xtrace	Generates a Paraver trace for a job's execution
xreservation	Manages reservations of resources

Resource requirements.

AEM uses the `Resources` structure which is embedded in the `JobDescription` to obtain the job's resource requirements. Specifically, the following `Resources` structures are supported:

- `OperatingSystemName` — Operating System name.
- `CPUArchitectureName` — Processor instruction set.
- `IndividualCPUSpeed` — Processor clock speed.
- `IndividualCPUCount` — Number of physical processors (CPU cores).
- `IndividualPhysicalMemory` — Physical RAM size.

Figure 4.10 shows a sample JSDL containing the resource requirements.

4.3.2 Using AEM through the command line

There is a set of command line utilities that give the user the ability of controlling the lifecycle of a job in the grid. Table 4.1 describes those commands briefly.

xsub

xsub xcommand provides a way to submit and execute a job providing a JSDL.

xsub [-h] [-V] [-v] [-c cert] [-r rev] -f jsdl

-h, –help Display this help and exit

-V, –version Output version information and exit

-v, –verbose Turn on verbose mode

-c, –cert=$\langle arg \rangle$ The filepath for user certificate. Default in /.xos/XATICAConfig.conf

-r, –rvid=$\langle arg \rangle$ Use the reservation ID specified for the execution. Defaults to using automatic reservations

```
<?xml version="1.0" encoding="UTF-8"?>
<JobDefinition xmlns="http://www.example.org/"
  xmlns:jsdl="http://schemas.ggf.org/jsdl/2005/11/jsdl"
  xmlns:jsdl-posix="http://schemas.ggf.org/jsdl/2005/11/jsdl-posix"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <JobDescription>
    <JobIdentification>
      <JobName>A demanding job</JobName>
      <Description>
        A sample resource requirements JSDL.
      </Description>
    </JobIdentification>
    <Application>
      <!-- an application description here -->
    </Application>
    <Resources>
      <IndividualPhysicalMemory>
        <Range>
          <LowerBound>1000002000.0</LowerBound>
          <UpperBound>2500000000.0</UpperBound>
        </Range>
      </IndividualPhysicalMemory>
      <IndividualCPUCount>
        <Exact>1.0</Exact>
      </IndividualCPUCount>
      <CPUArchitecture>
        <CPUArchitectureName>x86</CPUArchitectureName>
      </CPUArchitecture>
    </Resources>
  </JobDescription>
</JobDefinition>
```

Figure 4.10: A sample JSDL for limiting the resource requirements for the job.

-f, –jsdl=⟨*arg*⟩ Definition of the job in the JSDL format

Note that in a XtreemOS system there is also the possibility of directly executing a JSDL file with default options by just making it executable.

```
chmod u+x myjob.jsdl
./myjob.jsdl
```

xsub.sh

xsub.sh is a script that generates a JSDL file for the specified executable file. Afterwards it calls xsub to submit it.

xsub.sh ⟨*executable*⟩[*parameters*][−*in*⟨*input*⟩][−*out*⟨*output*⟩][−*err*⟨*error*⟩]

-in ⟨*input*⟩ Standard input for the job

-out ⟨*output*⟩ Standard output for the job

-err ⟨*error*⟩ Standard error for the job

xps

xps provides an interface to the monitoring infrastructure of XtreemOS. It's possible to select a specific job by its ID, a group of jobs with a tagged dependence or every job owned by the user identified in the certificate. A user is allowed to see basic monitoring information of any job in its VO.

xps [-h] [-V] [-v] [-c cert] [-A] ⟨[−*a*][−*T* − *jjobID*][−*jjobID*]⟩

-h, –help Display this help and exit

-V, –version Output version information and exit

-v, –verbose Turn on verbose mode

-c, –cert=⟨*arg*⟩ The filepath for user certificate. Default in /.xos/XATICAConfig.conf

-A, –ANSI Turn on ANSI for color presentation

-a, –all Show information for every job of the user

-j, –jobID=⟨*arg*⟩ Show information for the specified job

-T, –TAG=⟨*arg*⟩ Follow dependences down from the specified job and show information for those jobs as well.

Example output from xps

```
$ xps -a
fe0a5555-2bb5-4649-b9f2-c7e8778e7a46 @ 1245242806838 :
               jobID = fe0a5555-2bb5-4649-b9f2-c7e8778e7a46
               userDN = 8903e091-efc4-4c97-9053-d784ccc21b06
               VO = fcc1d2eb-b534-44d3-8daf-3b94c57b5035
               jobStatus = Done
               submitTime = Wed Jun 17 14:46:44 CEST 2009
004260af-91d4-4af5-853e-d01a54d0e67b @ 1245242806883 :
               jobID = 004260af-91d4-4af5-853e-d01a54d0e67b
               userDN = 8903e091-efc4-4c97-9053-d784ccc21b06
               VO = fcc1d2eb-b534-44d3-8daf-3b94c57b5035
               jobStatus = Running
               submitTime = Wed Jun 17 14:46:35 CEST 2009
               runNode = 84.88.50.161:60000
               PID = 10506
               processStatus = S
               userTime = 00:00.00
               systemTime = 00:00.00
               PID = 10505
               processStatus = S
               userTime = 00:00.00
               systemTime = 00:00.00
```

xkill

xkill provides a way to send events to jobs providing the jobID and the number of the event (or signal).

xkill [-h] [-V] [-v] [-c cert] -e event -j jobID

-h, –help Display this help and exit

-V, –version Output version information and exit

-v, –verbose Turn on verbose mode

-c, –cert=⟨*arg*⟩ The filepath for user certificate. Default in /.xos/XATICAConfig.conf

-j, –jobID=⟨*arg*⟩ JobID of the job receiving the signal

-e, –event=⟨*arg*⟩ The signal number that will be sent to the job

xwait

xwait will block until the jobID is done/finished, and prints the exit value.

xwait [-h] [-V] [-v] [-c cert] -j jobID

-h, –help Display this help and exit

-V, –version Output version information and exit

-v, –verbose Turn on verbose mode

-c, –cert=⟨*arg*⟩ The filepath for user certificate. Default in /.xos/XATICAConfig.conf

-j, –jobID=⟨*arg*⟩ JobID of the job to wait for

xreservation

xreservation xcommand provides a way to create and remove reservations.

xreservation

-a ⟨*reservationID*⟩ Output all the resources of the reservation

-1 ⟨*reservationID*⟩ Output first resource reserved (Internal Use)

-m Output to MPIRun special format [File MyResource] (Internal Use)

-n, ⟨*numResources*⟩ Number of resources that we will reserve

-t ⟨*durationInMinutes*⟩ Duration of the reservation

-f ⟨*arg*⟩ Definition of the job in the JSDL format, used for resource restriction

-z ⟨*reservationID*⟩ Removes the reservation

-X Exclusive reservation [default Mutual/Shared]

-C ⟨*CPUquantity*⟩ Quantity of CPU to reserve [1-100], default 1

-e Create an empty reservation and print the reservation ID of the newly created reservation

-L List the IDs of my reservations

-qr ⟨**reservationID**⟩ Print the details of the reservation

-qf Output the list free slots on accessable nodes. Optionally, use -T to specify a target node.

-T ⟨**comm.Address**⟩ Specify a target node. This is used optionally in conjunction of -qf, and in combination with -M, -add

-M ⟨**reservationID**⟩ modify a reservation. Needs to be used in conjunction with operation (-add).

-add an operation for modification of a reservation. It creates a request for adding a new allocation slot to an existing reservation. It needs to be used with target node specification (-T), time interval specification -time, and, optionally, with -X.

-time ⟨**start time**⟩ **-** ⟨**end time**⟩ define the time interval, in the format `dd.MM.yyyy hh:mm:ss`.

The easiest usage of the command is to let the scheduler select the reservation slots by passing the JSDL file, specifying the number of requested resources and the duration of the job.

```
 $ xreservation -n 2 -t 10 -f tmp.jsdl
 690dc3c6-1931-4d62-a76a-acd5ffe11d79
```

This command asked for 2 resources to be used by the job specified in tmp.jsdl, for duration of 10 minutes. It returned the ID of the reservation, which can be used as the value of the -r parameter of the **xsub** command.

To examine the details of the reservation, call the following:

```
 $ xreservation -qr 690dc3c6-1931-4d62-a76a-acd5ffe11d79
Address = [://172.16.118.193:60000(172.16.118.193)] \
                          291d7b0c-67e2-4bd4-85e5-dbdbcab6da5a
       30.06.2009 14:02:44 - 30.06.2009 14:12:44
Address = [://172.16.117.196:60000(172.16.118.193)] \
                          3e290a0b-7c70-4a4c-ac5f-955bea893eb1
       30.06.2009 14:02:44 - 30.06.2009 14:12:44
```

We can see that we have a reservation on nodes 172.16.118.193 and 172.16.117.196, concurrently for duration of 10 minutes. The right column next denotes the local allocation unit ID on each of the nodes.

Alternatively, we can manually request the time slots on the desired nodes. We start by first creating an empty reservation in order to obtain a reservation ID.

```
 $ xreservation -e
19572ce6-0cea-4425-a350-324efd217149
```

Then, we request a time slot on the node with address 172.16.118.193 to start at 6pm, and the job will take 75 minutes. We request an exclusive reservation for the time.

```
$ xreservation -M 19572ce6-0cea-4425-a350-324efd217149 \
        -add -T 172.16.118.193:60000 \
        -time 30.06.2009 18:00:00 - 30.06.2009 19:15:00 -X
```

If we would like to check which resources are available for requesting reservations, we invoke:

```
$ xreservation -qf
Address = [://172.16.117.196:60000]: \
               30.06.2009 14:02:44 : 30.06.2009 14:12:44

Address = [://172.16.118.193:60000]: \
               30.06.2009 14:02:44 : 30.06.2009 19:15:00
    30.06.2009 14:12:44 - 30.06.2009 18:00:00
```

The output shows the time when the node is not occupied by an existing reservation. The first line means that the node 172.16.117.196 can be used before 14:02:44, and after 14:12:44. The time within this interval represents pre-booked items that cannot be used. Similarily, the node 172.16.118.193 is booked starting with 14:12:44, and ending with 19:15:00. However, the lines that follow tell which intervals are free to take additional reservation requests. In this case, we could request a time slot any time between 14:12:44 and 18:00:00.

4.3.3 Using AEM through XConsole

The **xconsole_dixi** is a part of the XATI package. The utility uses a user certificate for the calls that need user's credentials. By default, it uses the path stored as **userCertificateFile** in the **~/.xos/XATIConfig.conf** configuration file. To override this value, it is also possible to use the command-line parameter **-c** *path_to_file.pem*.

Once in the XConsole shell, the following commands are available:

- `xsub -f jobdefinition.jsdl` — Job submission. It receives a jsdl file name as parameter. The xsub command creates and runs the job. It waits until the job id is returned and prints it.
- `xps -l` — Shows the info for the last submitted job.
- `xps -j jobID` — Shows the info for the job with id jobID.
- `xps -r [://IP:Port]` — Shows the info for jobs currently running on the node, defaults to the node where the command is executed.
- `xps -a` — Shows the info for all the submitted jobs.
- `xwait jobID` — It waits for the finalizations of a specific job specified by jobID.
- `xkill event jobID` — Sends the specified event (UNIX event) to the jobID.
- `xrs -jsdl jsdl_file` — Show the list of resources that fit into the jsdl requirements.
- `xcheckpoint jobID` — Job checkpointing.
- `xrestart jobID checkpointVersion` — Job restart.

4.3.4 Advanced features of AEM

In the following subsections some advanced features of AEM will be described, and some examples given to achieve the most typical objectives as reported by our users.

Running a job across different nodes using XATI

The XtreemOS Application Toolkit Interface (XATI) provides a way to communicate with the services of a core node through its exported methods. For the AEM component, the most useful methods for the user are present in the Job Manager service.

One of them is createProcess, a fork-like method to create and schedule a new process in the job. Note that if you just made a fork, the new process would be locally spawned. On the other hand, with createProcess, the process will be scheduled across the job's reservation in a Round-Robin fashion.

Let's follow an example, suppose you want to create a job composed by N processes distributed along N nodes. These would be the necessary steps to follow, example code in Figure 4.12:

1. Create a JSDL for the job and specify the number of nodes you need (example in Figure 4.11).

2. Create a job with the automatic reservations feature, this will be the first process (*createJob* method).

3. Create N-1 processes using *createProcess* method and specifying the jobID returned by the previous step.

The service methods used in the process are the following, the user is encouraged to read its javadoc to get deeper information about its parameters and return values:

- `static public java.lang.String createJob(String __jsdlFile, Boolean __startJob, String __reservationID, X509Certificate __userCtx) throws Exception`

- `static public java.lang.Integer createProcess(String __jobId, String __JSDL, String __reservationId, CommunicationAddress __resource, X509Certificate __userCtx) throws Exception`

In both methods a null reservationID is allowed to use the automatic reservations feature. Alternatively, the reservation can be programatically requested if the user may want to tune its parameters, specially the duration, which by default is just 10 min..

Another advanced feature, useful only in very specific cases is the expression of the resource parameter in the createProcess method. This way a specific resource from the reservation may be selected avoiding the schedule mechanism. Keep into account that the system will check the resource is a valid option before execution. More information about this functionality is given in the subsection.

```
<?xml version="1.0" encoding="UTF-8"?>
<jsdl:JobDefinition
  xmlns:jsdl="http://schemas.ggf.org/jsdl/2005/10/jsdl"
  xmlns:jsdl-posix"http://schemas.ggf.org/jsdl/2005/06/jsdlposix">
<jsdl:JobDescription>
    <jsdl:JobIdentification>
        <jsdl:JobName>ls</jsdl:JobName>
        <jsdl:Description>Xtreemos</jsdl:Description>
        <jsdl:JobProject>XtreemOS</jsdl:JobProject>
    </jsdl:JobIdentification>
    <jsdl:Application>
        <jsdl-posix:POSIXApplication>
            <jsdl-posix:Executable>sleep</jsdl-posix:Executable>
            <jsdl-posix:Argument>300</jsdl-posix:Argument>
        </jsdl-posix:POSIXApplication>
    </jsdl:Application>
    <jsdl:Resources>
        <jsdl:TotalResourceCount>
            <jsdl:exact>2</jsdl:exact>
        </jsdl:TotalResourceCount>
    </jsdl:Resources>
</jsdl:JobDescription>
</jsdl:JobDefinition>
```

Figure 4.11: A sample JSDL for requesting a specific number of nodes reserved.

```
 import eu.xtreemos.xati.CXATIConfig;
 import eu.xtreemos.xati.API.XJobMng;
 import eu.xtreemos.xosd.utilities.security.VOMSCommonUtils;

[...]
//This is just a snippet, full code in package eu.xtreemos.xati.test

 CXATIConfig xatiConfig = new CXATIConfig(path_to_XATIConfig);
 X509Certificate certificate = VOMSCommonUtils
     .readCert(xatiConfig.userCertificateFile);

/*
 * Call createJob without specifying a reservation (it should create
 * one automatically with the resources specified in the JSDL)
 */
 jobID = XJobMng.createJob(readJSDL(jsdl), true, null, certificate);
 System.out.println("Job ID obtained:" + jobID);

/*
 * Call createProcess without specifying a reservation (It should
 * use the reservation attached to the jobID) JSDL can be changed to
 * call another application (but resources will no be changed)
 */

 try {
     XJobMng.createProcess(jobID, readJSDL(jsdl), null, null,
         certificate);
 } catch (Exception e) {
     System.out.println("Error spawning process: " + e.toString());
 }
 System.out.println("New process created and running" + jobID);
 System.out.println("Waiting the job to finish");
 XJobMng.jobWait(jobID, certificate);

 System.out.println("Job Finished. The reservation is automatically
 cleaned as this is an automatic reservation.");
 System.exit(0);
```

Figure 4.12: A sample Java snippet using XATI to generate multiple distributed processes for one job

Running an MPI application in XtreemOS

Running an MPI application is in a preliminary and beta status (Won't work fully until 0.2.8 AEM packages are released). In the future, more integrated support into MPIRun will be completed.

Installation Follow the next steps to install MPI support.

1) Configure, make and install mpich-1.2.7-p1 from source in the client node. Download from `http://www.mcs.anl.gov/research/projects/mpi/mpich1/download.html` then

```
tar xvzf mpich.tar.gz
cd mpich-1.2.7-p1
./configure -rsh=/usr/bin/xsubMPI -prefix=/usr/bin
make
make install
```

This will force to use xsubMPI as submitting tool. However this can be configured later if we use -rsh=rsh with the P4_RSHCOMMAND=/usr/bin/xsubMPI as environment variable.

2) Copy the XtreemOS scripts that act as a wrapper of MPIRun.

(a) Copy /usr/share/dixi/bin/XOS_mpirun to /usr/bin (this script can be run from user directory)

(b) Copy /usr/share/dixi/bin/mpirun.xtreemosmpi to /usr/bin [where scripts for mpich are installed]

(c) Copy /usr/share/dixi/bin/mpirun.xtreemosmpipg to /usr/bin

Execution There are some considerations in order to run a MPI executable inside XtreemOS.

1) Command to execute (one line): Please check that xreservation, xsubMPI, and XOS_mpirun are accessible in the master node.

```
EXTERNALRES="xreservation -f /etc/skel/sleep.jsdl -n <numProcs> \
-t <durationinMinutes> -m" P4_RSHCOMMAND=xsubMPI MPI_HOST=local \
./XOS_mpirun -np <numProcs>  -nolocal <EXECUTABLE>
```

/etc/skel/cal.jsdl should be substituted by a dummy jsdl where some resource restrictions can be used (like x64 or x86), numProcs by the number of resource we should reserve and use, and durationinMinutes will be the duration of the reservation.

2) In the XtreemFS volume (home directory of the grid user) should contain .xos/ directory with the configuration for XATICA (would be near the same as the normal user, except the route to the certificates) and the certificates to execute xsubMPI remotelly. The fields address.host and address.port need to be removed, as xsubMPI will be launched from an unknown node.

We also need the executable to run.

The script will create one file during its execution: MyResource containing needed shared information. In order to test it, we recommend using a typical rank printing MPI application. This file should be in the XtreemFS home, so the execution should be done using ssh-xos or inside XtreemFS volume.

Note: Compilation/configuration is needed as long as P4_RSHCOMMAND doesn't work in some default installations. If this is the case, one solution is to symlink xsubMPI to rsh :

```
ln -s /usr/bin/xsubMPI /usr/bin/rsh
```

Creating several jobs inside the same reservation

To run several jobs inside the same reservation the user can proceed with a XATI program or use commands (xcommands).

The sequence of calls will be the next:

1. Create a reservation with *n* resources. Get the **reservationID**.
2. Call *n* times xsub (or createJob or createProcess) with this **reservationID** (parameter -r)

4.3.5 Running interactive jobs

The current support for interactive jobs in XtreemOS allows two basic types of interaction with applications running on the grid:

1. export user desktop environment to the application: the application can interact with the user *as if it was running on the desktop*.
2. allow the user to open interactive sessions inside a job context. This type of interaction allows the user to debug an application running on the Grid, for instance.

In case 1, the application can

- open X11 windows on the user desktop,
- get job control (signals, prompt) if run from a shell,
- sometimes use pipes (not yet fully supported in release 2.0),

```
<Application>
  <POSIXApplication
    xmlns:ns1="http://schemas.ggf.org/jsdl/2005/11/jsdl-posix">
    <Executable>/usr/bin/XOSJobDaemon</Executable>
    <Argument>--interactive</Argument>
    <Argument>/bin/bash</Argument>
    <Output>bash.out</Output>
    <Error>bash.err</Error>
  </POSIXApplication>
</Application>
<Resources>
  <TotalResourceCount>
    <Exact>1</Exact>
  </TotalResourceCount>
</Resources>
```

Figure 4.13: A jsdl for running a `bash` shell on the Grid.

- have input/output redirected to files local to the desktop.

Case 1 allows, for instance, a user to "run a bash shell on the Grid". This shell is subject to VO policies and resources used during the session are accounted as for a normal batch job. This shell is run with the user's Grid identity. The user can read/write/edit files from his XtreemFS home volume.

Case 2 mixes both the traditional batch job approach and the interactive job approach.

Interactive application

Figure 4.13 shows a part of a JSDL document which allows the user to submit a `/bin/bash` application to the Grid. This JSDL document requests AEM to start the `/usr/bin/XOSJobDaemon` service on the allocated node. This service forks the `/bin/bash` process when it has received the user environment. The first argument of `XOSJobDaemon` not starting with `--` if the the pathname of the application ro be executed interactively. This application is run using `xjob` as following:

```
[yvon@xos-core ~]$ xjob -T -X -f bash.jsdl
Job submitted succesfully: c38a3304-7129-4aed-86c3-31e6b61db3a4
Enter passphrase for key '/home/yvon/.xos/truststore/private/user.key':
bash-3.2$ id
uid=60000(/CN=fc1d3fa7-2373-474f-9f25-5301adf8a99a) gid=60443(xosuser_g60443) \
groups=60443(xosuser_g60443)
bash-3.2$ pwd
/home/fc1d3fa7-2373-474f-9f25-5301adf8a99a
bash-3.2$
```

The new user command `xjob` is based on command `xsub` extended with the capability to provide the user environment to the application. Argument `-T` requests `xjob` to forward the user `tty` to the application and argument `-X` to forward the user display. Forwarding a control tty allows line formatting (adding a prompt) and job control: typing `^ C` in the terminal does not kill the job, but is interpreted by the bash shell. Without argument `-T` (or with `-t`), the application simply gets raw input/output/err file descriptor and can be inserted into pipes. Command `xjob` is based on `ssh-xos` and `ssh-xos` needs to authenticate the user for interactive job submission. To avoid `ssh-xos` requesting the user key password for each job submission, it is possible to start a `ssh-xos` agent in charge of responding to key challenges.

In this example, the `/bin/bash` shell is executed under control of the execution manager on the resource node:

```
bash-3.2$ ps -ejHf
UID        PID  PPID  PGID   SID  C STIME TTY          TIME CMD
...
root      3369     1  2456  2421  0 17:35 ?        00:02:55   java -cp :/usr/share/dixi/libs/jrpcgen.jar:/us
root      4695  3369  2456  2421  0 17:35 ?        00:00:03     vmstat 3
root     20740  3369  2456  2421  0 23:07 ?        00:00:00     java -cp :/usr/share/dixi/libs/jrpcgen.jar:/
60000    20784 20740  2456  2421  0 23:07 ?        00:00:00       /usr/bin/XOSJobDaemon --interactive /bin/b
60000    20803 20784  2456  2421  0 23:07 ?        00:00:00         /usr/bin/XOSJobDaemon --interactive /bin
60000    20804 20803 20804 20804  0 23:07 pts/2    00:00:00           /bin/bash
60000    21380 20804 21380 20804  0 23:25 pts/2    00:00:00             ps -ejHf
...
```

Interactive job support allows users to edit files, prepare job and exploit their job results using their grid identity and while being subject to VO policies and resource accounting.

Interactive session inside application context

Figure 4.14 shows a part of a JSDL document which allows the user to first submit a job and then to run new commands inside the job context. This jsdl can be submitted using `xsub` or `xjob`:

```
[yvon@xos-core ~]$ xsub -f work.jsdl
Job submitted succesfully: 483d04ce-7f58-4a3d-aa06-85f53cf60ae6
```

Later, but before the job is terminated, the user can run commands inside the application context using `xjob`:

```
[yvon@xos-core ~]$ xjob -T -j 483d04ce-7f58-4a3d-aa06-85f53cf60ae6 /bin/bash
Enter passphrase for key '/home/yvon/.xos/truststore/private/user.key':
bash-3.2$ id
uid=60000(/CN=fc1d3fa7-2373-474f-9f25-5301adf8a99a) gid=60297(xosuser_g60297) \
groups=60297(xosuser_g60297)
```

```
<Application>
  <POSIXApplication
    xmlns:ns1="http://schemas.ggf.org/jsdl/2005/11/jsdl-posix">
    <Executable>/usr/bin/XOSJobDaemon</Executable>
    <Argument>--interactivesession</Argument>
    <Argument>/bin/sleep</Argument>
    <Argument>600</Argument>
    <Output>sleep.out</Output>
    <Error>sleep.err</Error>
  </POSIXApplication>
</Application>
<Resources>
 <TotalResourceCount>
   <Exact>1</Exact>
 </TotalResourceCount>
</Resources>
```

Figure 4.14: A jsdl for running new commands inside a job environment.

The process tree structure shows that the application (pid 21715) was started at job submission time (23:34) and later the /bin/bash shell was forked at 23:35 in the same context as the application:

```
bash-3.2$ ps -ejHf
UID        PID  PPID  PGID   SID  C STIME TTY          TIME CMD
...
root      3369     1  2456  2421  0 17:35 ?        00:03:02   java -cp :/usr/share/dixi/libs/jrpcgen.jar:/us
root      4695  3369  2456  2421  0 17:35 ?        00:00:03     vmstat 3
root     21670  3369  2456  2421  0 23:34 ?        00:00:00     java -cp :/usr/share/dixi/libs/jrpcgen.jar:/
60000    21706 21670  2456  2421  0 23:34 ?        00:00:00       /usr/bin/XOSJobDaemon --interactivesessio
60000    21715 21706  2456  2421  0 23:34 ?        00:00:00         /bin/sleep 600
60000    21783 21706  2456  2421  0 23:35 ?        00:00:00         /usr/bin/XOSJobDaemon --interactivesessi
60000    21784 21783 21784 21784  0 23:35 pts/2    00:00:00           /bin/bash
60000    21804 21784 21804 21784  0 23:36 pts/2    00:00:00             ps -ejHf
...
```

Note

The support for interactive job is experimental in XtreemOS 2.0. In the future, extensions to the jsdl syntax will allow to directly specify the execution environment of the application in the jsdl and so, to avoid explicitly referencing the XOSJobDaemon inside the jsdl. These extension will also allow xjob to detect when the control terminal and/or the desktop display needs to be forwarded to the application.

4.4 SRDS

The SRDS module is currently used by the AEM module. It will also export an interface toward XtreemOS applications, which is not available in the XtreemOS First Public Release.

4.5 XOSAGA

XOSAGA provides a high-level programming interface to XtreemOS. In this section we will develop a simple XOSAGA C++ application that mimics the 'cp' command. We will use this to easily copy files to and from XtreemFS volumes. More information and examples can be found in the programming guide:

`/usr/share/doc/libsaga-devel/saga_programming_guide.pdf`.

The source code of the example program looks like this:

```
#include <saga.hpp>
#include <iostream>

using namespace std;

int main(int argc, char** argv) {
    if (argc != 3) {
        cout << "usage: " << argv[0] << " srcFile dstFile" << endl;
        return 1;
    }

    saga::url srcUrl(argv[1]);
    saga::url dstUrl(argv[2]);

    try {
        saga::session s;
        saga::filesystem::file f(s, srcUrl, saga::filesystem::Read);

        cout << "Copying " << srcUrl << " to " << dstUrl << endl;
        f.copy(dstUrl);

    } catch (saga::exception const & e) {
        cout << "caught SAGA exception: " << e.get_message() << endl;
    }
}
```

It reads the source and destination file from the command line arguments, creates a SAGA file object and uses the copy() function to perform the copying. Save the program in a file called 'copyfile.cpp'.

Now create the following Makefile:

```
SAGA_SRC       = $(wildcard *.cpp)
SAGA_OBJ       = $(SAGA_SRC:%.cpp=%.o)
SAGA_BIN       = $(SAGA_SRC:%.cpp=%)

include $(SAGA_LOCATION)/share/saga/make/saga.application.mk
```

Compile your program:

```
$> make
      compiling  copyfile.o
      binlinking copyfile
$>
```

We can now use the copyfile program to copy files around. In SAGA, files are named using URLs. Normal, 'local' files can be accessed using just their normal file names, which is interpreted by SAGA as a URL that only contains a path name. Files on XtreemFS volumes are prefixed with 'xtreemfs://'. For example, a file 'foo.txt' on an XtreemFS volume 'vol42' located on a machine called 'storage.example.com' is referenced with the following URL:

`xtreemfs://vol42@storage.example.com/foo.txt`

SAGA takes care of mounting an unmounting the referenced volumes.

4.5.1 Testing the example application

The following example assumes you have an XtreemFS DS, MRC, and OSD process running locally. We will create a volume 'vol42', and copy a file to it using our example program.

Create the XtreemFS volume:

```
$> mkvol http://localhost/vol42
```

Create a file, and copy it to the volume:

```
$> echo 'howdy!' > foo.txt
$> ./copyfile foo.txt xtreemfs://vol42@localhost/bar.txt
```

You can check whether the file was copied succesfully by mounting the XtreemFS volume manually:

```
$> mkdir /tmp/vol42
$> xtreemfs -o volume_url=http://localhost:32636/vol42 -o direct_io \
-o logfile=/dev/null /tmp/vol42
$> ls /tmp/vol42
bar.txt*
$> cat /tmp/vol42/bar.txt
howdy!
$> fusermount -u /tmp/vol42
$> rmdir /tmp/vol42
```

Chapter 5

Application Examples

part will collect application examples ready to be run on a system that is assumed to be fully configured; the examples should cover all the basic services

Chapter 6

FAQ

Here goes a FAQ section with common errors a basic user can encounter, and either a concise solution or a reference to the admin guide for full discussion of the topic

6.1 Known Issues

6.2 FAQ

Bibliography

Appendix A

Public Resources on the Net

This appendix lists XtreemOS resources available on the Internet, for the common user, the System administrator and the developer.

A.1 Resources for Users

Table A.1: User Resources

Description	URL / contact info
Official WWW	http://www.xtreeemos.eu
XtreemOS Blog	https://www.xtreemos.org/blog
XtreemOS Mobile Guide	ftp://ftp.free.fr/mirrors/ftp.mandriva.com/MandrivaLinux/devel/xtreemos/2008.0/mobile/xtreemosmd-guide.pdf
IRC channel for user support	irc.freenode.netchannel #xtreemos

A.2 Code and Repositories

we should list here : the main Mandriva and Red Flag repositories for packages, Install cds, and the repositories where the Virtual Box /VMWare images are found;

A.3 Resources for Developers

important deliverables (or parts of them), references to some tech papers, tech tutorials?

Table A.2: Public Repositories

Description	URL / contact info
Mirror for CD Images	ftp://ftp.free.fr/mirrors/ftp.mandriva.com/MandrivaLinux/devel/iso/xtreemos/
XtreemOS Mobile distribution	ftp://ftp.free.fr/mirrors/ftp.mandriva.com/MandrivaLinux/devel/xtreemos/2008.0/mobile/
TikiWiki site	https://xtreemos.wiki.irisa.fr/
XtreemOS Development site	https://gforge.inria.fr/projects/xtreemos/
Public SVN	http://gforge.inria.fr/plugins/scmsvn/viewcvs.php/?root=xtreemos
Architecture documents	http://www.xtreemos.eu/publications/project-deliverables/

Table A.3: Developer Resources

Description	URL / contact info
IRC channel for user support	irc.freenode.netchannel #xtreemos-dev
TikiWiki site	https://xtreemos.wiki.irisa.fr/
Developer mailing list	http://lists.gforge.inria.fr/cgi-bin/mailman/listinfo/xtreemos-developers
XtreemOS Development site	https://gforge.inria.fr/projects/xtreemos/
Public SVN	http://gforge.inria.fr/plugins/scmsvn/viewcvs.php/?root=xtreemos
Architecture documents	http://www.xtreemos.eu/publications/project-deliverables/

Appendix B

Glossary

report acronyms and terms for the end-user

Index

This index right now is a tool to help editing; we shall consider producing real indexes, especially in the admin guide. This note appears on the wrong page