Project no. IST-033576

XtreemOS

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

User and Administrator's Guide

XtreemOS Technical Report # 5

Oscar David Sánchez

Report Registration Date: December 17, 2008

Version 1.03 / Last edited by Oscar David Sánchez / December 17, 2008

<table>
<tr><th colspan="3">Project co-funded by the European Commission within the Sixth Framework Programme</th></tr>
<tr><th colspan="3">Dissemination Level</th></tr>
<tr><td><b>PU</b></td><td>Public</td><td>√</td></tr>
<tr><td><b>PP</b></td><td>Restricted to other programme participants (including the Commission Services)</td><td></td></tr>
<tr><td><b>RE</b></td><td>Restricted to a group specified by the consortium (including the Commission Services)</td><td></td></tr>
<tr><td><b>CO</b></td><td>Confidential, only for members of the consortium (including the Commission Services)</td><td></td></tr>
</table>

Revision history:

Version	Date	Authors	Institution	Section affected, comments
0.1	20/05/08	Oscar David Sánchez		Initial document
0.2	09/06/08	Marko Novak	XLAB	Added text on how to use LinuxSSI PlugProPol framework
0.3	09/06/08	Marko Novak	XLAB	Added text on other LinuxSSI component (this is a copy-paste text from D2.2.7 and needs to be revised by other WP2.2 members)
0.4	10/06/08	Matej Artač, Aleš Černivec	XLAB	Added text on DIXI framefork and security services (RCA,VOPS)
0.5	04/07/08	Matthieu Fertré	INRIA	Updated text about LinuxSSI installation from XtreemOS Install CD
0.6	19/08/08	Luis Pablo Prieto	TID	Re-structuring of the document
0.7	08/09/08	Bjoern Kolbeck	ZIB	Modified and updated XtreemFS sections
0.8	09/09/08	Ian Johnson	STFC	Notes in Walkthrough section about certificates, description of CDA
0.8	16/09/08	Matthieu Fertré	INRIA	Updated text about LinuxSSI installation
0.85	02/10/08	Patrizio Dazzi	CNR	Added text about SRDS installation and configuration
0.85	03/10/08	Massimo Coppola	CNR	Added text about ADS_Bamboo installation and configuration
0.86	20/10/08	Ian Johnson	STFC	Expanded description in Walkthrough on the outline steps to install XtreemOS, noting the earliest stage in which the VO Management services can be started and tested.
0.87	28/10/08	Ian Johnson	STFC	Describe create-csr and process-csr. Uniform naming for keys and certificates. Checking user.crt against certificate chain.
0.90	30/10/08	Ian Johnson	STFC	Provide details in 'Localhost' section on how to install xtreemos-localhost, configure X-VOMS, start CDA server and use CDA client to get XOS-Certificate.
0.91	31/10/08	An Qin	ICT	Review the PAM-related sections, and provide the latest content in section.
0.92	3/11/08	Massimo Coppola	CNR	Reviewing the SRDS, RSS and ADS_Bamboo sections, chapters 4, 5.
0.93	03/11/08	Ian Johnson	STFC	Updated VOLife configuration - use volife.properties to find CDA key and certificate. Don't make dummy certificates for AEM.
0.94	04/11/08	Patrizio Dazzi	CNR	Updated SRDS, ADS_Bamboo and RSS configuration, added the SRDS client node section
0.95	06/11/08	An Qin	ICT	Updated localhost configuration, added the configuration of local policy setup
0.96	06/11/08	Aleš Černivec	XLAB	Refined text in 'Localhost' section - AEM/RCA/VOPS configuration, added cal.jsdl listing; updated listings of configuration AEM files to reflect latest changes.
0.97	12/11/08	Aleš Černivec	XLAB	Modifed section Configuring AEM for core and resource node set up.
0.98	20/11/08	Massimo Coppola	CNR	Clarifications about SRDS,RSS, ADS_Bamboo, in the core and resource configuration sections of the walkthrough.
0.99	28/11/08	Emanuele Carlini	CNR	Added informations on how running RSS recorder at core node configuration section. Minor changes at resource and client node configuration sections.
1.00	03/12/08	Oscar David Sánchez	INRIA	Completed missing sections, first final version of the documentation
1.01	05/12/08	Ramon Nou	BSC	Added PAM configuration for AEM
1.02	17/12/08	Oscar David Sánchez	INRIA	Additional commands in the localhost procedure to avoid known bugs
1.03	17/12/08	Ramon Nou	BSC	XtreemFS servers need to be restarted instead of just started on localhost

Contents

1 Introduction

1.1 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
  - Submit jobs
  - Check jobs and wait for finalization
  - Control jobs
  - Send events to jobs
  - Check resources that match job requirements
  - No global job scheduler
  - Distributed management of jobs
- Virtual organization management
  - 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 operation via web and command-line interfaces Operation via command line available
- Data management
  - Complete file system functionality
  - POSIX compliant
  - Grid authentication via XOS certificates
  - Volume creation and mount

  - Striping
  - Monitoring

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.

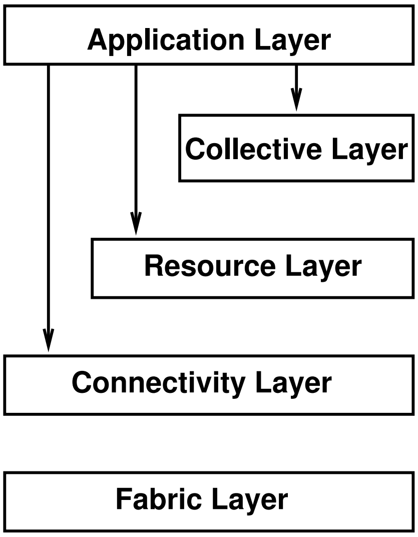

Figure 1: The layered Grid middleware architecture.

Not only does XtreemOS cover the whole spectrum of OGSA layers. XtreemOS also integrates operating systems for the various computer architectures used in VOs:

- For stand-alone PCs (single CPU, or SMP, or multi-core), XtreemOS provides its Linux-XOS flavour with full VO support.
- For clusters of Linux machines, the LinuxSSI flavour combines VO support with a single system image (SSI) functionality.
- For mobile devices, finally, XtreemOS provides the XtreemOS-MD flavour with VO support and specially-tailored, lightweight services for application execution, common data access, and user management.

1.2 The Guide

This Administrator's and User's guide provides comprehensive guidelines for installing, configuring and operating an XtreemOS system.

- Section 2 describes how an XtreemOS grid looks like, its architecture and the core, resource and client services that must be deployed in order to operate the system
- Section 3 explains how to obtain the latest XtreemOS install CD and its software packages
- Section 4 explains how to run XtreemOS in just one machine, so you can get familiar with the main concepts and operations.
- Section 5 explains how to set up a minimal XtreemOS grid in a fast and easy way
- Section 6 explains how to install and configure each XtreemOS service, in order to set up a full operative XtreemOS grid
- Section 7 describes how to operate an XtreemOS grid, its commands and the most common use cases

2 Overview

2.1 Architecture

There are 3 different kinds of nodes (also called *configurations*) in an XtreemOS system:

- 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.

However, this is just a logical (and functional) division, and nothing precludes a certain machine from acting as e.g. both a client node and a resource node. However, it is not advised to allow access on core nodes to users other than system administrators.

The different purpose of each configuration also determines which components and services should be present on it. In the following, each of these services are briefly described.

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. VOLifeCycle functionalities can still be accessed through a command-line interface.

Credential Distribution Authority (CDA) server is used to get an XOS-Certificate for the user, containing their public key and their VO attributes. The CDA server is accessed by the standalone 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 Bamboo DHT that is 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 access the CDA server in order to get user XOS-Certificates.

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 application certificates from the XtreemOS root CA. Application 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 application 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 application certificates from system administrators. This root certificate has to be installed on every machine in the XtreemOS Grid, to enable checking of received application 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 XOS certificate is `$HOME/.xos/truststore/certs/user.crt`, with the corresponding private key in `$HOME/.xos/truststore/private/user.key`.

- 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`. `<service>` can be `cda`, `vops`, `rca`.
- The storage of resource certificates is handled transparently by the AEM RCA client.

Table 1: XtreemOS Certificates

Certificate Type	Used By	Purpose	Created By
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 - see section 6.1
XOS certificate	End-user	Contains public key and VO attributes.	CDA server - see section 7.1.1
Application certificate	System admin	Used to authenticate the identity of applications running on core nodes, or to authenticate an XtreemFS client).	Root CA - see section 6.2.6
Service certificates	System admin	Used in similar manner to application certificates	RCA
Resource certificate	AEM	Attest to the resources provided by a resource node	RCA Server - see section 6.2.5

2.3 Installation and configuration process

The installation and configuration process of a XtreemOS grid should follow the following order (please refer to section 5 for minimal setup routine, or to section 6 for a more detailed view of the process):

1. Install XtreemOS and configure the Root CA on on a single machine (see section 6.1). This should not be a multiuser machine, and ideally should be physically secure. his machine should, ideally, not run any other XtreemOS services. To provide the ultimate in security (and 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 6.2.1).
   - VOLife (section 6.2.3).
   - CDA (section 6.2.2).

   This step requires using the Root CA to generate application certificates for the services (see section 7.1.2).

   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 6.7)

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

- SRDS (section 6.8).

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 6.2.5).
    - VOPS (secction 6.2.7).

5. Install and configure XtreemFS servers (section 6.9).

3 Getting XtreemOS

3.1 Install CD

This First release of an XtreemOS Linux distribution is based on the Mandriva Linux 2008.0 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.

3.2 Downloading the Install CD

The XtreemOS Install CDs are available from Mandriva mirrors. Several mirrors are available, you can find a list on this page : http://wiki.mandriva.com/en/XtreemOS

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

If you already have installed a Mandriva 2008.0 Linux distribution on your machine(s), you can make your machines VO-aware by installing the XtreemOS software packages which are available on the mirrors. See the next section about setting up packages repositories to install the packages.

3.4 Setting up packages repositories

Once your XtreemOS machine is installed, you may want to stay up to date by getting any bugfixes and new features that the XtreemOS community releases.

Doing this is easy. Select the mirror you want to use and run this command as root (replacing MIRRORURL by the URL of the mirror you selected) :

```
# urpmi.addmedia --distrib MIRRORURL/MandrivaLinux/devel/xtreemos/i586
```

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/
```

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

4 Setting up a single-PC installation of XtreemOS in 10 minutes

This chapter explains how to set up an installation of XtreemOS in just one machine. Of course, this is not a real grid at all, but it will give you the opportunity of getting familiar with the main concepts, functionalities and components of XtreemOS, without the need for a full grid infrastructure and just in a few minutes.

You just need to follow these steps:

1. Install XtreemOS from the install CD as a usual Linux installation
2. During installation, select the Xtreemos localhost installation option
3. Alternatively if you didn't select the localhost option during install, you can later install the `xtreemos-localconfig` package after setting up the packages repository (see section 3.4):

   ```
   # urpmi xtreemos-localconfig
   ```

4. Configure the X-VOMS database

   ```
   # /usr/share/xvoms/bin/xvoms_prepare_database.sh
   ```

5. Start the CDA server

   ```
   # /sbin/service cdaserver start
   ```

   This accesses the X-VOMS database.

6. Use the CDA client to contact the CDA server

   This is the simplest way of testing that the X-VOMS database has been set up correctly. The CDA client creates a private key and retrieves an example XOS-Certificate from the CDA server. To create these credentials for the pre-configured user, specify the location for the private key and certificate, and specify the 'test' flag:

   ```
   $ get-xos-cert localhost:6730 vo1 vouser.key vouser.crt test
   ```

Test mode supplies the username (’xtreemos-vouser’) and password (’xtreemos’) for the pre-configured user. The private key is stored in `vouser.key`, protected by the pass-phrase ’xtreemos’, and the XOS-Certificate is stored in `vouser.crt`. The name of the VO that the user wants to consider their ’primary VO’ is specified by the second argument to the command - in this case, the X-VOMS database contains just one VO, vo1.

7. View the XOS-Certificate (long lines below have been wrapped as indicated by the backslash character ’)́:

```
$ view-xos-cert vouser.crt
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            01:1e:21:f7:bd:f4
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: O=CDA for localhost config, OU=cda, CN=localhost/cda
        Validity
            Not Before: Dec 10 17:30:30 2008 GMT
            Not After : Jan  9 17:40:30 2009 GMT
        Subject: CN=ea9a7366-e34f-4a99-9e31-277430366475
        X509v3 extensions:
            X509v3 Basic Constraints: critical
                CA:FALSE
            X509v3 Key Usage: critical
                Digital Signature, Key Encipherment
            X509v3 Extended Key Usage: critical
                TLS Web Client Authentication
            X509v3 Authority Key Identifier:
                keyid:9D:D7:3E:20:84:CC:0E:3F:62:85:C1:6F:\
                32:0D:66:4A:06:F2:8C:73
                DirName:/CN=XtreemOS CA/O=XtreemOS Project\
                /OU=XtreemOS Project Root Certification Authority
                serial:01

            X509v3 Subject Key Identifier:
                85:FC:6E:89:5A:C8:30:D4:3D:4D:75:0D:EC:C6:83:\
                25:42:66:5B:6A
        XtreemOS VO Attributes:

            GlobalPrimaryVOName:
                2c0e8cb2-4453-46fe-85b7-74874e76e7c2
            GlobalSecondaryVONames:
                null
            GlobalUserID:
                ea9a7366-e34f-4a99-9e31-277430366475
            GlobalPrimaryGroupName:
                ae88816f-9f5c-48f9-ad7d-f71a64977904
            GlobalSecondaryGroupNames:
                null
            Role:
                null
            Group:
                group1
            Subgroup:
                null
```

Further tests of the VO Management service, such as creating a VO and VO groups/roles, are described later in this document: see section 7.1, 'Using VOLife'.

In this step, you can know the VO name of the already generated user certificate. This name will be used in the next steps. In this localhost configuration, the user's VO name is *2c0e8cb2-4453-46fe-85b7-74874e76e7c2*. You can find the VO name under **GlobalPrimaryVOName**.

8. Setup the local policy

   Most of the XtreemOS configuration files are stored in /etc/xos, and certificates are stored in /etc/xos/truststore. In particular, local policy configuration file is /etc/xos/nss_pam/pam_xos.conf. Modify it so we indicate the Certification Authority certificate and the certificates path (last two lines):

```
# emacs /etc/xos/nss_pam/pam_xos.conf
VOCACertDir       /etc/xos/truststore/certs
VOCACertFile      xtreemos.crt
```

Then, you can run the amsd daemon and setup the local policy via following commands:

```
# /sbin/service xos-amsd start
# xos-policy-admin-am -vo 2c0e8cb2-4453-46fe-85b7-74874e76e7c2 --force
# xos-policy-admin-gm -vo 2c0e8cb2-4453-46fe-85b7-74874e76e7c2 --force
```

As a test of the local policy mapping, issue the following command:

```
# pam_app_conv -pem vouser.crt
```

You should see output similar to the following:

```
vo = [2c0e8cb2-4453-46fe-85b7-74874e76e7c2], role = [null]
Server running with PID: 11555
[/CN=ea9a7366-e34f-4a99-9e31-277430366475@your-host tmp]$
```

This has changed your effective user identity (effective UID) to a UID managed by the local mapping policy, and started a new shell running in the /tmp directory.

If this works correctly, you can exit this shell (by typing CTRL-D) and continue with the next steps.

9. Start XtreemFS services

   In order to use XtreemFS, you must restart its services: the Directory Service, the MRC and the OSD:

   ```
   $ /sbin/service xtreemfs-dir restart
   $ /sbin/service xtreemfs-mrc restart
   $ /sbin/service xtreemfs-osd restart
   ```

   To check that these services are running, open your web browser and direct it to http://localhost:32638, http://localhost:32636 and http://localhost:32640, where you will find the status page of the Directory Service, the MRC and the OSD respectively.

10. Create and mount and XtreemFS volume

    ```
    $ xtfs_mkvol -p RAID0,256,1 http://localhost/myVolume
    $ mkdir ~/xtreemfs
    $ xtfs_mount -o volume_url=http://localhost/myVolume ~/xtreemfs
    ```

Further operations of XtreemFS are described later in this document: see section 7.6, 'Using XtreemFS'.

11. Set up RSS

    Then, you need to provide your IP address (and network interface and or disk device, if needed) to start the Resource Selection Service. To do that, edit the RSS configuration file and enter your IP in the bootstrap_address field.

    ```
    $ emacs /etc/xos/config/Rss/config.cfg
    network_interface = eth0
    disk_device = sda4
    bootstrap_address = 131.254.201.25
    ```

12. Set up AEM and RCA server and client

    In this step we will need again the VO name of the already generated user certificate. See step 6 if needed. AEM and RCA need several certificates that have been already installed by the `xtreemos-localconfig` package.

    Now you can start the xosd service, that will run all the services needed to execute jobs locally (you need to have root privileges):

    ```
    $ sudo service xosd start
    ```

    Then, you can ask RCA

    ```
    $ rca_apply
    Returned from service call: successMethod
    $ rca_list_pending
    ```

    Last command should list pending resources to be added to RCA database. In the following lines ⟨*vo name*⟩ and ⟨*machine IP*⟩ should be the same as returned by the **rca_list_pending** command.

    ```
    $ rca_confirm <machine IP>:60000
    Returned from service call: successMethod
    $ rca_request
    Returned from service call: successMethod
    $ sudo chmod 777 /etc/xos/truststore/certs/incoming/
    $ dixi_test -RCA avo <vo name> <machine IP>:60000
    ```

```
Returned from service call: successMethod
Adding the resource 10.0.2.15:60000 to the VO.
Added resource 10.0.2.15:60000 to \
VO 2c0e8cb2-4453-46fe-85b7-74874e76e7c2.
$ sudo cp /etc/xos/truststore/certs/incoming/\
attrcert<GlobalPrimaryVOName>ext.crt \
/etc/xos/truststore/certs/
```

IP used in upper commands can be different than yours. Do not forget to replace ⟨*GlobalPrimaryVOName*⟩ and ⟨*machine IP*⟩ with *2c0e8cb2-4453-46fe-85b7-74874e76e7c2* (obtained with step 7) and IP of the machine running AEM. With **rca_apply** this node applies for joining VO. With **rca_confirm** machine's joining is confirmed. Command **rca_request** requests for obtaining VO attribute certificate and with last three commands newly generated certificate is placed in expected directory.

13. Test AEM

With these steps job submission using AEM is tested. In order to do this, user certificates, which were generated in step 6, have to be copied under specified directory (if paths are not specified with **get-xos-cert** command already).

XtreemOS user directories are under /.xos. Certificates and some client configuration files are stored there. Verify that the following directories exist. Otherwise, you must create them in your home folder:

```
$ mkdir ~/.xos
$ mkdir ~/.xos/truststore
$ mkdir ~/.xos/truststore/certs
$ mkdir ~/.xos/truststore/private
```

```
$ cp vouser.crt ~/.xos/truststore/certs/user.crt
$ cp vouser.key ~/.xos/truststore/private/user.key
```

Job description file's content (*/etc/skel/cal.jsdl*), which will be used in a job submission test, is as follows:

```
<JobDefinition xmlns="http://schemas.ggf.org/jsdl/2005/11/jsdl">
    <JobDescription>
        <JobIdentification>
```

```
              <Description>Execution of cal</Description>
              <JobProject>XtreemOS_Test</JobProject>
          </JobIdentification>
          <Application>
             <POSIXApplication
xmlns="http://schemas.ggf.org/jsdl/2005/11/jsdl-posix">
                  <Executable>/usr/bin/cal</Executable>
                  <Output>/tmp/out_cal.txt</Output>
                  <Error>/tmp/err_cal.txt</Error>
              </POSIXApplication>
          </Application>
      </JobDescription>
 </JobDefinition>
```

Jsdl file must have right read permissions:

```
$ chmod 755 /etc/skel/cal.jsdl
```

After executing next commands, you should obtain following result (*I*P and *u*ser can be different).

```
[user@localhost ]$ xconsole_dixi
XtreemOS Console
$ xrs -a
Listing all resources:
  Address = [://10.0.2.15/10.0.2.15:60000(10.0.2.15/10.0.2.15)]
$ xsub -f /etc/skel/cal.jsdl
Job submitted successfully: ed1a36fa-57ea-42ce-9502-16aadfde62a5
```

At this point we have successfully set up AEM locally and job submission should be enabled. We will find the output of the job in /tmp/out_cal.txt.

5 Setting up your XtreemOS grid on a Testbed composed of multiple PCs

This chapter explains how to set up a small XtreemOS testbed. The idea is to set up an XtreemOS grid with every needed service in few nodes, starting with the XtreemOS install CD. That is not so much for a grid, but it will show how to install and configure all the modules needed in a VO. Adding resources and users afterwards should be quite straightforward. The installation process is divided in three parts: the Core node one, the Resource node one and the Client node one. The activation of the SSL with the AEM will be shown in the section 6.5.

5.1 Setting up a Core node

5.1.1 Setting up the packages repositories

After installing from the XtreemOS CD, you may need to update the packages from the online repositories. See section 3.4 about how to setup packages repositories and update your system.

5.1.2 Configuring the certificates

A Virtual Organization needs a Root Certification Authority and its public certificate. This subsubsection describes how private and public Root CA certificates can be generated.

On the machine which will be running the Root CA, install the rootca-config package :

```
(root)# urpmi rootca-config
(user)$ ls /etc/xos/config/openssl/
create-rootca-creds.conf  process-csr.conf
```

NB You do not need to create the root CA under the Linux 'root' account, but you will need root permissions to install some of the files which are created in the following steps. These steps are indicated by the shell prompt 'Root #' preceding the command.

The certificates will be stored in the directory you choose, for example, `/opt/xtreemosca`:

```
$ init-rootca /opt/xtreemosca
$ ls /opt/xtreemosca
certs/  index.txt  private/  public/  serial
```

The next step is to create the Root CA private key and self-signed public key certificate. This step is required when creating the Root CA, and when the public key certificate expires (with the default settings, every 365 days).

The new Root CA certificate should be distributed before the current one expires. The OpenSSL configuration is defined in the file `/etc/xos/config/openssl/create-rootca-creds.conf`. The section `[ root_ca_distinguished_name ]` can be modified to change the certificate fields `commonName`, `organizationName` and `organizationalUnitName` as required.

```
[ root_ca_distinguished_name ]
commonName = XtreemOS CA
organizationName = XtreemOS Project
organizationalUnitName = XtreemOS Project Root Certification Authority
```

The next command creates the root CA private key and public key certificate. You will be prompted for a passphrase - this protects the private key, and is required when using the Root CA to create server certificates from Certificate Signing Requests (CSRs).

```
$ create-rootca-creds /opt/xtreemosca
$ ls /opt/xtreemosca/private/
xtreemos.key
$ ls /opt/xtreemosca/public/
xtreemos.crt
```

The CA public key certificate must be in the XtreemOS truststore directory:

```
Root # cp /opt/xtreemosca/public/xtreemos.crt \
      /etc/xos/truststore/certs/
```

The public key certificate of the Root CA is the XtreemOS Root Certificate. It needs to be installed on all machines in this XtreemOS Grid.

In all the examples that follow, the hostname `'host'` should be replaced by either the Fully-Qualified Domain Name for the host (its DNS entry), or the IP address of the host (the latter should be seen only as a temporary measure).

The package create-csr contains instructions and an OpenSSL configuration file to create a certificate signing request (CSR) file for an application (client or service).

Install the create-csr package on any machine where you wish to make a CSR file (it need not be on the same machine that is running the Root CA).

This is used to obtain application certificates for the core services (or XtreemFS client).

```
$ create-csr host "My Organization" cda
$ ls
host-cda.csr
host-cda.key
```

Now you need the Certification Authority to sign the host-cda.csr request.

```
$ process-csr /opt/xtreemosca host-cda.csr
Using configuration from /etc/xos/config/openssl/process-csr.conf
Enter pass phrase for /opt/xtreemosca/private/xtreemos.key:********
...

$ ls
host-cda.crt
host-cda.csr
host-cda.key

$ openssl x509 -text -in host-cda.crt -noout
Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number:
            9c:11:53:54:5e:11:e0:83
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: CN=XtreemOS CA, O=XtreemOS Project, \
                OU=XtreemOS Project Root Certification Authority
        Validity
            Not Before: Sep 17 08:30:59 2008 GMT
            Not After : Sep 17 08:30:59 2009 GMT
        Subject: CN=host/cda, O=My Organization, \
                 OU=cda
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:ce:d9:fe:50:51:f7:c4:f4:bf:49:69:4b:1a:44:
                    ...
                    0c:66:dc:3f:13:63:7e:d8:eb
                Exponent: 65537 (0x10001)
    Signature Algorithm: sha1WithRSAEncryption
        2f:fc:8c:9c:6f:4d:97:27:1d:f2:0d:0e:11:a4:50:0b:c2:1a:
        ...
```

It is possible to create other application certificates (for RCA, VOPS, etc), following the procedure above, changing the last argument passed to `create-csr` to `rca` or `vops`, and operating the Root CA to create certificates from the CSR files.

You have to put the CDA Server, RCA Server and VOPS application certificates in the directory `/etc/xos/trustore/certs/` on this machine. For the CDA application, the procedure is:

```
Root # certdir=/etc/xos/truststore/certs # short-cut to reduce typing
Root # cp host-cda.crt ${certdir}/cda.crt
Root # chmod a+r ${certdir}/cda.crt # Anyone can read the public key
Root # chown cdauser.cdauser ${certdir}/cda.crt
```

The corresponding private key must be placed in `/etc/xos/trustore/private`. For the CDA application, the procedure is:

```
Root # keydir=/etc/xos/truststore/private # short-cut to reduce typing
Root # cp host-cda.key ${keydir}/cda.key
Root # chmod a+r ${keydir}/cda.key
#
# users 'cdauser' and 'tomcat' need to read the key
# File permissions are reduced, but key is pass-phrase protected
#
Root # chown cdauser.cdauser ${keydir}/cda.key
# Key is owned by the 'service user' e.g. 'cdauser' for CDA
```

Now you need to link the certificates with their hash. The OpenSSL command `c_rehash` will create hash files for all the certificates in a directory:

```
c_rehash /etc/xos/truststore/certs
```

A hash file is created for each of the certificates in the directory. The 'stem' of the filename is a hash of the certififcate contents, the suffix is either '.0' or '.1' in case of a hashing collision. The `c_rehash` command needs to be run on the directory whenever certificates are added to it.

You need to configure PAM to use the Root CA certificate:

```
Root # cat /etc/xos/nss_pam/pam_xos.conf
VOCACertDir            /etc/xos/truststore/certs
VOCACertFile           xtreemos.crt
```

V̲OLife certificate configuration

When generating XOS-Certs, VOLife uses settings in `/etc/xos/config/volife/volife.properties` to locate the CDA private key and certificate, and to supply the pass-phrase protecting the private key. These settings can be copied from the CDA configuration in `/etc/xos/config/cdaserver/cdaserver.properties`.

```
$ cat /etc/xos/config/volife/volife.properties
cdaserver.keyFilename=/etc/xos/truststore/private/cda.key
cdaserver.keyPassphrase=changeme
cdaserver.certFilename=/etc/xos/truststore/certs/cda.crt
```

The key pass-phrase should be changed to the actual pass-phrase protecting the private key.

5.1.3 Configuring XtreemFS

You do not need to configure any of the XtreemFS services for a single-node setup. UUIDs are automatically assigned to the services during installation.

The default setup will work *without SSL*. To enable SSL, get service certificates for the Directory Service, the MRC, the OSD and the client. See The XtreemFS User Guide, Section "Configuring SSL Support".

```
Root # service xtreemfs-dir restart
stoping XtreemFS Directory Service (DIR)...                     [  OK  ]
starting XtreemFS Directory Service (DIR)...                    [  OK  ]
Root # service xtreemfs-mrc restart
```

```
stoping XtreemFS Metadata and Replica Catalog (MRC)...          [  OK  ]
starting XtreemFS Metadata and Replica Catalog (MRC)...         [  OK  ]
Root # service xtreemfs-osd restart
stoping XtreemFS Object Storage Device (OSD)...                 [  OK  ]
starting Object Storage Device (OSD)...                     [  OK  ]
```

5.1.4 Configuring VOLife

Ensure that VOLife is configured to find the CDA private key and certificate as described in section 5.1.2 above. Then configure the X-VOMS database, if this hasn't already been done:

```
Root # /usr/share/xvoms/bin/xvoms_prepare_database.sh
```

Now configure VOLife to access the XtreemFS services:

```
Root # cd /usr/share/tomcat5/webapps/volifecycle/WEB-INF/classes
Root # cat MRC.properties
mrc.host=localhost
mrc.port=32636
```

If the XtreemFS server is on an other host, mrc.host = IP of the XFS host.

You need to restart tomcat5 to run VOLife:

```
Root # service tomcat5 restart
```

There is a work-around for bug #6843:

```
Root # mkdir /usr/share/tomcat5/certs
Root # chown tomcat.tomcat /usr/share/tomcat5/certs
```

Get User XOS-Certificates with VOlife The VOLife permits you to configure VOs and Users. You can access it with a browser at this address: `http://host:8080/volifecycle`

The first step is the registration of the VO administrator. Sign up to the VOLife webapp:
`VOAdmin`
`password`
An XtreemFS volume is created for this user.

You are the VO administrator of the VO you create:
— `Create a VO: VOName - VVVV`.

If it is the first time. you have to create your user private key:
— `Generate new key pair`:
Click on `Generate` (you need to create a new password) and `Download` your private key - it is stored in the file `user.key`.

Now you need a XOS-Cert to authenticate to the chosen VO. Click on Get a XOS-Cert and choose the VO you have created: — `Get a XOS-Cert` for the VO `VVVV`.

You need to enter your private key password. Then you can download the public certificate by clicking on `Download`. You have now the `user.crt` file.

Create the directory `$HOME/.xos/trustore` directory with sub-directories `private` and `certs` and copy the private key and XOS-Certificate there:

```
$ mkdir -p $HOME/.xos/truststore/private/
$ mkdir -p $HOME/.xos/truststore/certs/
$ cp user.key $HOME/.xos/truststore/private/
$ cp user.crt $HOME/.xos/truststore/certs/
```

You can check that the XOS-Certificate `user.crt` can be verified against the certificate chain:

```
$ cd
$ openssl verify -CApath /etc/xos/truststore/certs \
.xos/truststore/certs/user.crt
.xos/truststore/certs/user.crt: OK
```

5.1.5 Configuring the local policy

Before using the PAM module, the Account Mapping Server (AMS) must be running:

```
Root # service xos-amsd restart
```

Now test your public certificate with the local policy tool, *xos-policy-admin-chk*:

```
Root# cd
Root# xos-policy-admin-chk -pem .xos/truststore/certs/user.crt
dn = [UUUU], vo = [VVVV], role = [null]
Sucess in PAM checking !
```

If you get the sucess message, the configuration of local policy is ready. The tool also presents the number of user's DN,VO,ROLE. But generally, there are not any policy are defined in the beginning, so you may get the following message usually:

```
Root # xos-policy-admin-chk -pem $HOME/.xos/truststore/certs/user.crt
dn = [UUUU], vo = [VVVV], role = [null]
Mapper: Unfound the match rule!! check your mapping rule,pls
PAM:fail in mapping connect !
        * a)Please check whether AMS daemon is running correctly *
        * b)Please check whether mapping rules are correct.      *
        *     If not, try:                                       *
        *       xos-policy-admin-am  -vo <vo> --force            *
        *       xos-policy-admin-gm  -vo <vo> --force            *
        * c)Please check whether setting rule is correct.        *
        *     If not, try:                                       *
        *       xos-policy-admin-set -uidmax <num> -uidmin <num> *
        *                            -gidmax <num> -gidmin <num> *
Oops: Permission denied
```

The *Mapper* in AMS outputs the message on missing mapping rules, so following the hint b) to add the mapping rules as local policy:

```
Root # xos-policy-admin-am  -vo VVVV  --force
Root # xos-policy-admin-gm  -vo VVVV  --force
```

Then, try again the above checking. If you got the following message from *Mapper*:

```
Root # xos-policy-admin-chk -pem $HOME/.xos/truststore/certs/user.crt
dn = [UUUU], vo = [VVVV], role = [null]
Mapper:Undefine the uid/gid spaces for mapping!! check your
setting rule,pls
PAM:fail in mapping connect !
...
```

You have to add another policy rule to told the AMS how large the space (span) local uid/gid is mapped in. By the tool, *xos-policy-admin-set*, it is eay to do that following the hint c):

```
Root # xos-policy-admin-set -uidmax 60500 -uidmin 60000 \
-gidmax 60500 -gidmin 60000
```

And then, try again the checking tool and the sucess message will be presented in screen. You can check the added rules by *xos-policy-admin-prt*.

```
Root # xos-policy-admin-prt
------------- rules database -----------------
[key:1]: NpSHoU8MwZakNPagJs3g7HulYXkQ0bn
Rule Type:Mapping rule
Subject:
Type: <DN,VO,ROLE>
Content_1: *
Content_2: VVVV
Object:
Type: local account
Content:*
Rule Content:
Driver Name:
Driver Params:
Valid: 1
TimeStamp: 1220521399

[key:2]: Gp3pVDW5SuCgKz4u6HZ1yHMkd1VBMVQ
Rule Type:Mapping rule
Subject:
Type: <VO,ROLE,ATTRS>
Content_1: VVVV
Content_2: *
Object:
Type: local groups
Content:*
Rule Content:
Driver Name:
Driver Params:
Valid: 1
TimeStamp: 1220521403

[key:-1]: GpistDW5Su7uh5fu6HZ1yHMkd1VBMVQ
uid space:
uid_max: 60500
```

```
uid_min: 60000
gid space:
gid_max: 60500
gid_min: 60000
----------------------------------------------
```

If the core node allows access users read/write global home volume, the automount option has to be opened after installation. Local administrator may configure the functionality by editing PAM's configuration file.

```
Root # vim /etc/xos/nss_pam/pam_xos.conf
...
OpenAutoMount yes
...
```

Opening the file and setting the option to "yes" will help user's home volume mount to their local home directories (in /home, named with their DN number).

NOTICE:

For mounting the global XtreemFS volume sucessfully, the user's home volume must already exist. Currently, the auto-mount mechanism will not create the corresponding home volume in XtreemFS. Manually creating the user's home volume in XtreemFS can be done by the following steps:

1) Make sure the /etc/xos/xtreemfs/default_dir has be specified the IP address and port of XtreemFS Directory Service (as an example, here are the details for *xtreemos1.zib.de*).

```
dir_service.host = xtreemos1.zib.de
dir_service.port = 32638
```

2) Create a home volume in XtreemFS.

```
Root # xtfs_mkvol -p RAID0,128,1 http://xtreemos1.zib.de/user-<UUUU>
```

Here, UUUU is the user's GUID taken from their XOS-Certificate (with the prefix '/CN=' removed). 3) Configure the fuse subsystem. Automount mechanism requires the some FUSE configuration to allow non-root to umount home directory. At default, the fuse is loaded. If not, load the fuse with the command:

```
Root # modprobe fuse
```

5.1.6 Configuring SSH-XOS

From a client machine we can connect to any core or resource machine.

```
Client root $ emacs /etc/ssh/ssh_config-xos
XosProxyFile    $YOUR_HOME/.xos/truststore/certs/user.crt
Quit emacs.

Core root $ emacs /etc/ssh/sshd_config-xos
UsePAM yes
Quit emacs.
```

Now you can connect the VO:

```
Client $ ssh-xos host
-bash-3.2$ whoami
/CN=c4b32574-cf06-47e7-b960-97e7f6b994a4
Ctrl-D.
```

5.1.7 Testing XtreemFS

XtreemFS is an object-based file system designed for federated IT infrastructures that are connected by wide-area networks.

To create your data volume:

```
$ xtfs_mkvol -a 2 http://localhost:32636/TestVolume
```

To see your volume :

```
$ xtfs_lsvol http://localhost:32636/
TestVolume  ->  00065BBD8E7C900B51481CF8
```

If necessary :

```
Root # modprobe fuse
```

Create a folder and mount the volume:

```
$ mkdir $HOME/TestVolume
Root # xtfs_mount -o dirservice=http://localhost, \
       volume_url=http://localhost:32636/TestVolume, \
       direct_io,allow_other $HOME/TestVolume
```

allow_other permits to access the data without being root.

Create a file in your volume:

```
$ touch $HOME/TestVolume/test.txt

$ ls $HOME/TestVolume
test.txt
```

to umount this volume :

```
Root # umount $HOME/TestVolume
$ ls $HOME/TestVolume
(nothing)
```

Mount it again :

```
Root # xtfs_mount -o http://localhost, \
       volume_url=http://localhost:32636/TestVolume, \
       direct_io,allow_other $HOME/TestVolume
$ ls $HOME/TestVolume
test.txt

Root # umount $HOME/TestVolume
$ xtfs_rmvol http:/localhost:32636/TestVolume
```

5.1.8 Configuring SRDS

Configuring SRDS and its dependencies (RSS and ADS_Bamboo) means editing some files to add proper IP and port numbers. The files defining the configuration are:

- /etc/xos/config/Ads/ADSConfig.xml,
- /etc/xos/config/Rss/config.cfg
- /etc/xos/config/Bamboo/stdconf.cfg

You only need to edit the second and third files, that is the RSS file and the Bamboo one.

In order to work correctly, RSS need the RSS recorder running only at one core node, named bootstrap node. If you are configuring core nodes, you have to choose which one will act as the RSS bootstrap. Note that, after this choice, all other core nodes and all resource and client nodes will have to get this very same IP for the Rss bootstrap.

In order to run the RSS recorder (tracker) on the machine at **bootstrap_address**, from that machine, go to the directory with XtreemOS jars (`/usr/share/java`) and run

```
java -cp DIXIMain.jar:srds.jar:xtreemrss.jar:log4j-1.2.14.jar \
eu.xtreemos.ads.Threads.RecorderRssThread
```

Rss file configuration:

- network_interface : the network interface to use (usually *eth0*, use *lo* for localhost testing)
- disk_device : the disk devide to monitor for free space
- bootstrap_address : IP address of bootstrap node for the RSS overlay

Bamboo stdconf.cfg file:

- in the “global” subsection, the “node_id” should be an *IP:port* couple; substitute the IP of the local node, or 127.0.0.1 for local testing, leave the port number to 3630.
- in the “Router” subsection, the “gateway” should be the IP (public) and port (3630) of the bootstrap node of Bamboo. If you are configuring core nodes, you have to choose which one will act as the Bamboo bootstrap, and put there its IP. Note that, after this choice, all other core nodes and all resource and client nodes will have to get this very same IP for the Bamboo bootstrap. (*When installing on a single machine, which is both core and resource, either use the local core IP address, or the couple 127.0.0.1:3630 for local testing with no network access).*

Last, configure the SRDS services within the Xosd, in order to enable them if they are not. Adding the SRDS as a service within the XOSD configuration can be done by editing the XOSdConfig.conf and inserting the following line:

```
services.13=eu.xtreemos.xosd.srdsmng.service.SRDSMngHandler
```

5.1.9 Configuring AEM

Configuration files of the AEM reside under /etc/xos/config. **XOSdConfig.conf** is a main configuration file and with it we can start other services by adding appropriate lines into array of services to be started. Other configuration files of services are JobDirectory.conf, JobMng.conf, RCAClientConf.conf, RCAServerConfig.conf, ResMng.conf, ResourceMonitorConfig.conf and VOPSConfig.conf. The existence of these depends of starting the service. Each service that is started with adding it into XOSdConfig can auto-generate appropriate configuration file.

In the following steps, please replace the example IP 131.254.201.16 with IP of your core machine, to adapt your configuration.

It is possible to generate the XOSd configuration files by running the XOSd server.

```
# service xosd start
# service xosd stop
```

This way auto-generated /etc/xos/conf/XOSdConfig.conf would differ from the one listed here (only subset of services are generated). In order to start core services on this node, we have to run appropriate services. We can do that by editing configuration In the **XOSdConfig**:

- **networkInterface** (leave if as it is if not sure),
- **rootaddress.host** is IP address of the Core node,
- **rootaddress.externalAddress** if the Core node is behind NAT (or set it equal to **rootaddress.host**, if not sure leave it as it is),
- if this node (the Core node) is behind NAT, change **externalAddress** or leave it as it is if not behind NAT (if not sure, just leave it as it is).
- edit (add lines) to **services** entries (listing below) and make sure that **services.size** always exceeds the numbers under **services** .

```
# emacs /etc/xos/config/XOSdConfig.conf

rootaddress.host=131.254.201.16
rootaddress.port=60000
rootaddress.externalAddress=131.254.201.16
```

```
externalAddress=131.254.201.16
networkInterface=eth0

useSSL=false
xmlport=55000
xosdRootDir=.

trustStore=/etc/xos/truststore/certs/aem_trusted/
trustStoreSSL=/etc/xos/truststore/certs/dixi_ssl/
privateKeyLocation=/etc/xos/truststore/private/reskey.key
certificateLocation=/etc/xos/truststore/certs/rescert.crt

services.size=15

services.13=eu.xtreemos.xosd.srdsmng.service.SRDSMngHandler
services.12=eu.xtreemos.xosd.jobmng.service.JobMngHandler
services.11=eu.xtreemos.xosd.security.vops.service.VOPSHandler
services.10=eu.xtreemos.xosd.security.rca.client.service.RCAClientHandler
services.7=eu.xtreemos.xosd.execMng.service.ExecMngHandler
services.5=eu.xtreemos.xosd.resmng.service.ResMngHandler
services.4=eu.xtreemos.xosd.jobDirectory.service.JobDirectoryHandler
services.3=eu.xtreemos.xosd.resourcemonitor.service.ResourceMonitorHandler
services.2=eu.xtreemos.xosd.xmlextractor.service.XMLExtractorHandler
services.1=eu.xtreemos.xosd.security.rca.server.service.RCAServerHandler
services.0=eu.xtreemos.xosd.daemon.DaemonGlobal

Quit emacs.
```

In the upper listing there are all services that need to be started in order to provide all core services of the AEM. Core node can be registered as a resource with executing the following commands:

```
Core node $ service xosd start
Core node $ rca_apply
Core node $ rca_list_pending
Core node $ rca_confirm 131.254.201.16:60000
Core node $ rca_request
```

If you are not sure about the IP to be used with command **rca_confirm**, **rca_list_pending** should list the IP to be used here.

5.1.10 Restarting and using AEM

If you restart the Core node, you have to check that the following services are running again:

XtreemFS service:

```
Root # service xtreemfs-dir restart
Root # service xtreemfs-mrc restart
Root # service xtreemfs-osd restart
```

Mysql service:

```
Root # service mysql restart
```

VOlife service:

```
Root # service tomcat5 restart
```

Account mapping service:

```
Root # service xos-amsd restart
```

Run the resource monitoring:

```
Root # service gmond start
```

And the XOSd service:

```
Root # service xosd start
```

If you need to kill the XOSd Server before restarting it:

```
Root # service xosd stop
Ctrl+C it once the XOSd is finished.
```

A resource need to be registered for running a job. See the Resource subsection.

For running a job, we need a user. See the Client subsection.

5.2 Setting up a Resource node

5.2.1 Setting up the packages repositories

After installing from the XtreemOS CD, you may need to update the packages from the online repositories. See section 3.4 about how to setup packages repositories and update your system.

5.2.2 Configuring the certificates

You need the Root CA certificate on your machine:

```
Root # cp xtreemos.crt /etc/xos/truststore/certs/
```

You need to use the Core machine public certificate to authenticate SSL access to the services on the core node(s). You can find all the certificates (`host-cda/rca/vops.crt)` in the `/etc/xos/truststore/certs/` folder of the core node(s). Copy them to this resource node.

You have to put the CDA Server, RCA Server and VOPS public certificates in the `/etc/xos/trustore/certs/` on this machine.

```
Root # cp *.crt /etc/xos/truststore/certs/
```

Now you need to link the certificates with their hash. The OpenSSL command `c_rehash` will create hash files for all the certificates in a directory:

```
c_rehash /etc/xos/truststore/certs
```

You need to configure PAM to use the Root CA certificate:

```
Root # emacs /etc/xos/nss_pam/pam_xos.conf
VOCACertDir            /etc/xos/truststore/certs
VOCACertFile           xtreemos.crt
Quit emacs.
```

5.2.3 Get User XOS-Certificates with VOLife

The VOLife permits you to register and to join a VO. You can access it with a browser at this address: `http://server_machine_ip:8080/volifecycle`

— Create a Resource owner user:
`Resource`
`password`
An Xtreemfs Volume is created for the user.

You are not the VO administrator of the VO you want to Join.
Click on Join a VO then choose the right VO and click on the `JoinVO` button for the VO `VOName - VVVV`

You have to wait that the VOAdmin adds you at the VO.

If it is the first time you have to create your user private key:
— `Generate new key pair`: Click on `Generate` (you need to create a new password) and `Download` you private key (user.key).

Now you need a XOS-Cert to authenticate to the chosen VO. Click on Get a XOS-Cert and choose the VO you have created:
— `Get a XOS-Cert` for the VO called VOName with the VO ID `VVVV`.

You need to enter your private key password. Don't forget to download the public certificate by clicking on `Download`. You have now the `user.crt` file.

Create the directories `$HOME/.xos/truststore/{private,certs}` folders and and copy the files `user.crt` and `user.key` there:

```
$ mkdir -p $HOME/.xos/truststore/private/
$ mkdir -p $HOME/.xos/truststore/certs/
$ cp user.key $HOME/.xos/truststore/private/
$ cp user.crt $HOME/.xos/truststore/certs/
```

You can verify the user certificate against the chain of certificates used in its creation. This assumes that the root CA certificate and the CDA certificate are installed in `/etc/xos/truststore/certs`, and that `c_rehash` has been run on the directory.

```
$ cd
$ openssl verify -CApath /etc/xos/truststore/certs/ .xos/truststore/certs/use
.xos/truststore/certs/user.crt: OK
```

5.2.4 Configuring the local policy

Before using the PAM module, the Account Mapping Server (AMS) must be running:

```
Root # service xos-amsd restart
```

Now test your public certificate with the local policy tool, *xos-policy-admin-chk*:

```
Root # xos-policy-admin-chk -pem $HOME/.xos/truststore/certs/user.crt
dn = [UUUU], vo = [VVVV], role = [null]
Sucess in PAM checking !
```

If you get the sucess message, the configuration of local policy is ready. The tool also presents the number of user's DN,VO,ROLE. But generally, there are not any policy are defined in the beginning, so you may get the following message usually:

```
Root # xos-policy-admin-chk -pem $HOME/.xos/truststore/certs/user.crt
dn = [UUUU], vo = [VVVV], role = [null]
Mapper: Unfound the match rule!! check your mapping rule,pls
PAM:fail in mapping connect !
        * a)Please check whether AMS daemon is running correctly *
        * b)Please check whether mapping rules are correct.      *
        *     If not, try:                                       *
        *       xos-policy-admin-am  -vo <vo> --force            *
        *       xos-policy-admin-gm  -vo <vo> --force            *
        * c)Please check whether setting rule is correct.        *
        *     If not, try:                                       *
        *       xos-policy-admin-set -uidmax <num> -uidmin <num> *
        *                            -gidmax <num> -gidmin <num> *
Oops: Permission denied
```

The *Mapper* in AMS outputs the message on missing mapping rules, so following the hint b) to add the mapping rules as local policy:

```
Root # xos-policy-admin-am  -vo VVVV  --force
Root # xos-policy-admin-gm  -vo VVVV  --force
```

Then, try again the above checking. If you got the following message from *Mapper*:

```
Root # xos-policy-admin-chk -pem $HOME/.xos/truststore/certs/user.crt
dn = [UUUU], vo = [VVVV], role = [null]
Mapper:Undefine the uid/gid spaces for mapping!! check your
setting rule,pls
PAM:fail in mapping connect !
...
```

You have to add another policy rule to told the AMS how large the space (span) local uid/gid is mapped. By the tool, *xos-policy-admin-set*, it is eay to do that following the hint c):

```
Root # xos-policy-admin-set -uidmax 60500 -uidmin 60000 \
-gidmax 60500 -gidmin 60000
```

And then, try again the checking tool and the sucess message will be presented in screen. You can check the added rules by *xos-policy-admin-prt.*

If the resource node allows access users read/write global home volume, the automount option has to be opened after installation. Local administrator may configure the functionality by editing PAM's configuration file.

```
Root # vim /etc/xos/nss_pam/pam_xos.conf
...
OpenAutoMount yes
...
```

Opening the file and setting the option to ”yes” will help user's home volume mount to their local home directories (in /home, named with their DN number).

Retry it with *pam_app_conv* which can open a session,mounting with global XtreemFS home volume.

```
Root # pam_app_conv -pem $HOME/.xos/truststore/certs/user.crt
```

If all is allright, you enter a new shell :

```
[11:44:28] core_machine /home/c4b32574-cf06-47e7-b960-97e7f6b994a4
/CN=c4b32574-cf06-47e7-b960-97e7f6b994a4 $
...
Ctrl-D.
```

5.2.5 Configuring the SRDS and RSS

Configuring SRDS and its dependencies (RSS and ADS_Bamboo) means to edit some files with proper IP and port numbers. In particular, the files defining the configuration are:

- /etc/xos/config/Ads/ADSConfig.xml,
- /etc/xos/config/Rss/config.cfg
- /etc/xos/config/Bamboo/stdconf.cfg

only the latter two are required to be edited: the Rss configuration file and the Bamboo one.

Rss configuration:

- network_interface : the network interface to use (likely *eth0*, use *lo* for localhost testing)
- disk_device : the disk device to monitor for free space
- bootstrap_address : the bootstrap node for the RSS overlay (the IP of the node chosen as Core Node, the local IP address or 127.0.0.1 for localhost testing)

Note: the boostrap node is the core node with Rss recorder running at.

SRDS configuration is limited to the Bamboo stdconf.cfg file:

- in the "global" subsection, the "node_id" should be an *IP:port* couple; substitute the IP of the local node, or 127.0.0.1 for local testing, leave the port number to 3630.
- in the "Router" subsection, the "gateway" should be the IP (public) and port (3630) of the bootstrap node of Bamboo. Put here your Core Node IP address, the local core IP address for a single machine, or the couple 127.0.0.1:3630 for local testing.

Last, configure the SRDS services within the Xosd, in order to enable them if they are not. Adding the SRDS as a service within the XOSD configuration can be done by editing the XOSdConfig.conf and inserting the following line:

```
services.13=eu.xtreemos.xosd.srdsmng.service.SRDSMngHandler
```

In the following section the configuration for the monitoring service within the AEM is reported, instead of the one accessing the SRDS.

5.2.6 Configuring the AEM

In my system the Resource IP is 131.254.201.21 and the Core and Root XOSd IP is 131.254.201.16. You need to adapt your configuration under /etc/xos/config/XOSdConfig.conf in order to tell this node (the Resource node) where the Core node is. First start and stop the XOSd service:

```
Root # service xosd start
Root # service xosd stop
```

This should auto-generate three configuration files under /etc/xos/config:

- XOSdConfig.conf
- ResourceMonitorConfig.conf
- RCAClientConfig.conf

Edit newly generated XOSdConfig.conf (note that configuration file's entries are not ordered in the same order as the listing below) and correct IPs appropriately. Edit next lines:

- **networkInterface** (leave if as it is if not sure),
- **rootaddress.host** is IP address of the Core node,
- **rootaddress.externalAddress** if the Core node is behind NAT (or set it equal to **rootaddress.host**, if not sure leave it as it is),
- if this node (the Resource node) is behind NAT, change **externalAddress** or leave it as it is if not behind NAT (if not sure, just leave it as it is).

```
Root # emacs /etc/xos/config/XOSdConfig.conf
rootaddress.host=131.254.201.16
rootaddress.port=60000
rootaddress.externalAddress=131.254.201.16
externalAddress=131.254.201.21
networkInterface=eth0

useSSL=false
xmlport=55000
```

```
xosdRootDir=.

trustStore=/etc/xos/truststore/certs/aem_trusted/
trustStoreSSL=/etc/xos/truststore/certs/dixi_ssl/
privateKeyLocation=/etc/xos/truststore/private/reskey.pem
certificateLocation=/etc/xos/truststore/certs/rescert.pem

services.size=15
services.8=eu.xtreemos.xosd.security.rca.client.service.RCAClientHandler
services.9=eu.xtreemos.xosd.execMng.service.ExecMngHandler
services.10=eu.xtreemos.xosd.resourcemonitor.service.ResourceMonitorHandler
services.11=eu.xtreemos.xosd.xmlextractor.service.XMLExtractorHandler
services.12=eu.xtreemos.xosd.daemon.DaemonGlobal

Quit emacs.
```

Note that numbers in **services** entries can be different as listed above.

Now you need to generate your client configuration files. You could run xconsole_dixi or xsub to generate them:

```
$ xconsole_dixi
```

Now modify them:

- **networkInterface** (leave if as it is if not sure),
- **userKeyFile** is the path to the user's private key,
- **userCertificateFile** is the path to the user's public certificate,
- **xosdaddress.host** IP address of the XOSd running on the Resource node,
- **xosdaddress.externalAddress** if the Resource node is behind NAT (or set it equal to **xosdaddress.host**, if not sure leave it as it is),
- if this node (the Resource node) is behind NAT, change **externalAddress** or leave it as it is if not behind NAT (if not sure, just leave it as it is).

If running **xconsole_dixi**:

```
$ emacs $HOME/.xos/XATIConfig.conf

#Properties File for the client application
#Mon Nov 10 20:39:57 CET 2008
useSSL=false
xosdaddress.externalAddress=131.254.201.21
xosdaddress.host=131.254.201.21
privateKeyLocation=/etc/xos/truststore/private/xati_dummy.key
userKeyFile=$YOUR_HOME/.xos/truststore/private/user.key
networkInterface=eth0
trustStoreSSL=/etc/xos/truststore/certs/dixi_ssl/
address.host=131.254.201.21
userCertificateFile=$YOUR_HOME/.xos/truststore/certs/user.crt
xosdaddress.port=60000
address.port=10000
certificateLocation=/etc/xos/truststore/certs/xati_dummy.crt

Quit emacs.
```

If running **xsub**: Here you should modify:

- **useSSL** set it to false,
- **xosdaddress.host** IP address of the XOSd running on the Resource node,
- **address.host** IP address of the XATI running on the Resource node (same as **xosdaddress.host**),
- **cdaaddress** IP address of the XOSd running on the Core node with running CDA service (if not sure, leave it as it is or set it to the IP of this node).

```
$ emacs $HOME/.xos/XATICAConfig.conf

useSSL=false
certificateLocation=/etc/xos/truststore/certs/xati_dummy.pem
privateKeyLocation=/etc/xos/truststore/private/xati_dummy.pem
trustStoreSSL=/etc/xos/truststore/certs/dixi_ssl/

xosdaddress.host=131.254.201.21
xosdaddress.port=55000
address.host=131.254.201.21
address.port=10000
cdaaddress.host=131.254.201.16
cdaaddress.port=60000

Quit emacs.
```

5.2.7 Add a resource with the AEM

Run the resource monitoring:

```
Root # service gmond start
```

(for killing the XOSd if necessary)

```
Root # service xosd stop
```

If you are on the Core node then just run xosd. If not the Root XOSd must be running on 131.254.201.16 Core node. Link your Resource XOSd to Core XOSd:

```
Root # service xosd start
```

1- The Core node needs to collect the details on the Resource node to be registered and to send the details and the node's ID to the RCA server for enlisting the resource to the resources pending for registration:

```
Resource $ rca_apply

Resource $ rca_list_pending
Returned from service call: successMethod
Listing pending resources:
ResouceID = [IP=131.254.201.21:60000]: [hostIP={Address =
[://131.254.201.21/131.254.201.21:60000(/131.254.201.21)]},
hostUniqueID={131.254.201.21}, operatingSystemName={Linux},
processorArchitecture={x86}, CPUCount={1.0}, RAMSize={2.125463552E9}]

Resource $ rca_list_registered
Returned from service call: successMethod
Listing pending resources:
  List empty.
```

2- When the resource does rca_confirm a higher instance on the Core node confirms the registration of the resource. This is probably done by an administrator at the organisation. So, on the Core node:

```
Core node $ rca_confirm 131.254.201.21:60000

Resource $ rca_list_pending
Returned from service call: successMethod
Listing pending resources:
  List empty.

Resource $ rca_list_registered
Returned from service call: successMethod
Listing registered resources:
ResouceID = [IP=131.254.201.21:60000]: [hostIP={Address =
[://131.254.201.21/131.254.201.21:60000(/131.254.201.21)]},
hostUniqueID={131.254.201.21}, operatingSystemName={Linux},
processorArchitecture={x86}, CPUCount={1.0}, RAMSize={2.125463552E9}]
```

It creates or update the RCA database in the file pointed by rcaDBFile in the RCAServerConfig.conf config file of the Core node.

```
Core node $ ls /etc/xos/RCADB.bin
/etc/xos/RCADB.bin
```

3- The Core node can create a key pair and request the signature of the RCA server with:

```
Resource $ rca_request

Requesting a new certificate...
Identity certificate:
 ...
```

4- The Core node generates a attributes certificates for the Resource node. You need the VO ID and the Resource node IP:

Before all, you have to open the incoming directory on your host:

```
Resource node $ chmod 777 /etc/xos/truststore/certs/incoming/

Core node $ dixi_test -RCA avo 7485601c-43b0-413f-83a5-a968b1835aea \
                        131.254.201.21:60000
```

So the Resource is registered to the VO and you get a attribute certificate in:

```
Resource node $ ls /etc/xos/truststore/certs/incoming/
attrcert7485601c-43b0-413f-83a5-a968b1835aeaext.pem

Resource node:
Root # cp /etc/xos/truststore/certs/incoming/attrcert7485601c...1835aeaext.pem \
  /etc/xos/truststore/certs/
```

This node is now available as a VO 7485601c-43b0-413f-83a5-a968b1835aea resource. We can display the registered resources with xconsole_dixi with the blank.jsdl file.

```
$ emacs $HOME/blank.jsdl

<?xml version="1.0" encoding="UTF-8"?>
<JobDefinition xmlns:jsdl="http://schemas.ggf.org/jsdl/2005/11/jsdl">
   <JobDescription>
       <JobIdentification>
             <Description>Blank</Description>
             <JobProject>Blank</JobProject>
         </JobIdentification>
         <Application>
             <POSIXApplication
                  xmlns:ns1="http://schemas.ggf.org/jsdl/2005/11/jsdl-posix">
                 <Executable></Executable>
             </POSIXApplication>
         </Application>
     </JobDescription>
</JobDefinition>

Quit emacs.

$ xconsole_dixi
XtreemOS Console
```

```
$xrs -jsdl $HOME/blank.jsdl
Listing resources matching JSDL query:
  Address = [://131.254.201.21/131.254.201.21:60000(/131.254.201.21)]

Pay attention to not put a space between the prompt and the xrs in the xconso
```

It is OK to run a job on this resource.

5.3 Setting up a Client node

5.3.1 Setting up the packages repositories

After installing from the XtreemOS CD, you may need to update the packages from the online repositories. See section 3.4 about how to setup packages repositories and update your system.

5.3.2 Configuration certificates

You need the Root CA certificate on your machine:

```
Root # cp xtreemos.crt /etc/xos/truststore/certs/
```

You need to use the Core machine public certificate to access the services on these machine. You can find all the certificates {`cda,rca,vops`}`.crt` in the directory `/etc/xos/truststore/certs/` of the core server machines.

You have to put the public certificates in the directory `/etc/xos/trustore/certs/` on this machine:

```
Root # cp corehost-{cda/rca/vops}.crt /etc/xos/truststore/certs/
```

Now you need to link the certificates with their hash:

```
c_rehash /etc/xos/truststore/certs
```

You need to configure PAM to use the Root CA certificate:

```
Root # emacs /etc/xos/nss_pam/pam_xos.conf
VOCACertDir            /etc/xos/truststore/certs
VOCACertFile           xtreemos.crt
Quit emacs.
```

5.3.3 Get User XOS-Certificates with VOLife

The VOLife permits you to register and to join a VO. You can access it with a browser at this address: `http://server_machine_ip:8080/volifecycle`

— Create a Client user:
`Client`
`password`
An Xtreemfs Volume is created for the user.

You are not the VO administrator of the VO you want to Join. Click on Join a VO then choose the right VO and click on the `JoinVO` button for the VO `VOName - VVVV`

You have to wait that the `VOAdmin` adds you at the VO.

If it is the first time you have to create your user private key:
— `Generate new key pair`:
Click on `Generate` (you need to create a new password) and `Download` your private key (user.key).

Now you need a XOS-Cert to authenticate to the chosen VO. Click on Get a XOS-Cert and choose the VO you have created:
— `Get a XOS-Cert` for the VO called VOName with the VO ID `VVVV`.

You need to enter your private key password. Don't forget to download the public certificate by clicking on `Download`. You have now the `user.crt` file.

Create the user .xos folder and the certificate subfolder and copy the `user.crt` and `user.key` certificates in the `$HOME/.xos/truststore/{private,certs}` folder:

```
$ mkdir -p $HOME/.xos/truststore/private/
$ mkdir -p $HOME/.xos/truststore/certs/
$ cp user.key $HOME/.xos/truststore/private/
$ cp user.crt $HOME/.xos/truststore/certs/
```

You can check that the XOS-Certificate can be verified against the certificate chain:

```
$ cd
$ openssl verify -CApath /etc/xos/truststore/certs/ \
.xos/truststore/certs/user.crt
.xos/truststore/certs/user.crt: OK
```

5.3.4 Configuring the local policy

Before using the PAM module, the Account Mapping Server (AMS) must be running:

```
Root # service xos-amsd restart
```

Now test your public certificate with the local policy tool, *xos-policy-admin-chk*:

```
Root # xos-policy-admin-chk -pem $HOME/.xos/truststore/certs/user.crt
dn = [UUUU], vo = [VVVV], role = [null]
Sucess in PAM checking !
```

If you get the sucess message, the configuration of local policy is ready. The tool also presents the number of user's DN,VO,ROLE. But generally, there are not any policy are defined in the beginning, so you may get the following message usually:

```
Root # xos-policy-admin-chk -pem $HOME/.xos/truststore/certs/user.crt
dn = [UUUU], vo = [VVVV], role = [null]
Mapper: Unfound the match rule!! check your mapping rule,pls
PAM:fail in mapping connect !
        * a)Please check whether AMS daemon is running correctly *
        * b)Please check whether mapping rules are correct.      *
        *     If not, try:                                       *
        *       xos-policy-admin-am  -vo <vo> --force            *
        *       xos-policy-admin-gm  -vo <vo> --force            *
        * c)Please check whether setting rule is correct.        *
        *     If not, try:                                       *
        *       xos-policy-admin-set -uidmax <num> -uidmin <num> *
        *                            -gidmax <num> -gidmin <num> *
Oops: Permission denied
```

The *Mapper* in AMS outputs the message on missing mapping rules, so following the hint b) to add the mapping rules as local policy:

```
Root # xos-policy-admin-am  -vo VVVV  --force
Root # xos-policy-admin-gm  -vo VVVV  --force
```

Then, try again the above checking. If you got the following message from *Mapper*:

```
Root # xos-policy-admin-chk -pem $HOME/.xos/truststore/certs/user.crt
dn = [UUUU], vo = [VVVV], role = [null]
Mapper:Undefine the uid/gid spaces for mapping!! check your
setting rule,pls
PAM:fail in mapping connect !
...
```

You have to add another policy rule to told the AMS how large the space (span) local uid/gid is mapped. By the tool, *xos-policy-admin-set*, it is eay to do that following the hint c):

```
Root # xos-policy-admin-set -uidmax 60500 -uidmin 60000 \
-gidmax 60500 -gidmin 60000
```

And then, try again the checking tool and the sucess message will be presented in screen. You can check the added rules by *xos-policy-admin-prt*.

If the client node allows access users read/write global home volume, the automount option has to be opened after installation. Local administrator may configure the functionality by editing PAM's configuration file.

```
Root # vim /etc/xos/nss_pam/pam_xos.conf
...
OpenAutoMount yes
...
```

Opening the file and setting the option to "yes" will help user's home volume mount to their local home directories (in /home, named with their DN number).

Retry if necessary :

```
Root # pam_app_conv -pem $HOME/.xos/truststore/certs/user.crt
```

If all is allright, you enter a new shell :

```
[11:44:28] paraxos /tmp
/CN=c4b32574-cf06-47e7-b960-97e7f6b994a4 $
Ctrl-D.
```

Remember your CN, it is your ID number : c4b32574-cf06-47e7-b960-97e7f6b994a4

5.3.5 Configuring SSH-XOS

Try to connect to the VO Core machine.

```
Client root $ emacs /etc/ssh/ssh_config-xos
XosProxyFile    $YOUR_HOME/.xos/truststore/certs/user.crt
Quit emacs.

Core root $ emacs /etc/ssh/sshd_config-xos
UsePAM yes
Quit emacs.
```

(NB You should substitute $YOUR_HOME for the pathname of your home directory.)

Connect to the destination machine:

```
Client $ ssh-xos core_machine
-bash-3.2$ whoami
/CN=c4b32574-cf06-47e7-b960-97e7f6b994a4
Ctrl-D.
```

5.3.6 Configuring the SRDS and RSS

Configuring SRDS and its dependencies (RSS and ADS_Bamboo) means editing some files to add proper IP and port numbers. The files defining the configuration are:

- /etc/xos/config/Ads/ADSConfig.xml,
- /etc/xos/config/Rss/config.cfg
- /etc/xos/config/Bamboo/stdconf.cfg

You only need to edit the second and third files, that is the Rss file and the Bamboo one.

Rss configuration:

- network_interface : the network interface to use (usually *eth0*, use *lo* for localhost testing)
- disk_device : the disk devide to monitor for free space

- bootstrap_address : the bootstrap node for the RSS overlay. All resource and client nodes will have to get the same IP for the Rss bootstrap that was chosen when configuring the core nodes.

Note: the boostrap node is the core node with Rss recorder running at.

Bamboo stdconf.cfg file:

- in the "global" subsection, the "node_id" should be an *IP:port* couple; substitute the IP of the local node, or 127.0.0.1 for local testing, leave the port number to 3630.
- in the "Router" subsection, the "gateway" should be the IP (public) and port (3630) of the bootstrap node of Bamboo. Put here your Core Node IP address, that was chosen when configuring the core node(s).

Last, configure the SRDS services within the Xosd, in order to enable them if they are not. Adding the SRDS as a service within the XOSD configuration can be done by editing the XOSdConfig.conf and inserting the following line:

```
services.13=eu.xtreemos.xosd.srdsmng.service.SRDSMngHandler
```

5.3.7 Configuring the AEM

You have to configure some files in /etc/xos/config/ and .xos/ folders.

In my system the Client IP is 131.254.201.20, the Resource IP is 131.254.201.21 and the Core and root xosd IP is 131.254.201.16. You need to adapt your configuration.

It is possible to generate the XOSd configuration files by running the XOSd server but it is difficult to stop it.

```
Root # service xosd start
Root # service xosd stop (or kill the XOSd PID)
```

Configuration files:

```
Root # emacs /etc/xos/config/XOSdConfig.conf
rootaddress.host=131.254.201.16
rootaddress.externalAddress=131.254.201.16
externalAddress=131.254.201.20
```

```
rootaddress.port=60000
xmlport=55000

xosdRootDir=.

useSSL=true
trustStore=/etc/xos/truststore/certs/aem_trusted/
trustStoreSSL=/etc/xos/truststore/certs/dixi_ssl/
privateKeyLocation=/etc/xos/truststore/private/reskey.pem
certificateLocation=/etc/xos/truststore/certs/rescert.pem

services.size=15
services.8=eu.xtreemos.xosd.security.rca.client.service.RCAClientHandler
services.9=eu.xtreemos.xosd.execMng.service.ExecMngHandler
services.10=eu.xtreemos.xosd.resourcemonitor.service.ResourceMonitorHandler
services.11=eu.xtreemos.xosd.xmlextractor.service.XMLExtractorHandler
services.12=eu.xtreemos.xosd.daemon.DaemonGlobal

Quit emacs.
```

Now the right services can be generated. So you can generate new configuration files with the XOSd:

```
Root # service xosd start
Root # service xosd stop

Root # emacs /etc/xos/config/RCAClientConfig.conf
cdaCertificateFileName=/etc/xos/truststore/certs/corehost-rcacert.pem
resIdentityCertFileName=/etc/xos/truststore/certs/rescert.pem
resPrivateKeyFileName=/etc/xos/truststore/private/reskey.pem
resAttributeCertExtFileName=/etc/xos/truststore/certs/attrextcert.pem
resAttributeCertFileName=/etc/xos/truststore/certs/attrcert.pem
resVOAttributeCertIncoming=/etc/xos/truststore/certs/incoming/

Root # emacs /etc/xos/config/ResMng.conf
VOPSPubCert=/etc/xos/truststore/certs/vops.crt
testVOPS=true
useADS=true

Root # emacs /etc/xos/config/ResourceMonitorConfig.conf
gangliaPort=8649
```

```
cpuVals.size=3
memVals.size=3
xMonMemProbe=mem_probe
xMonCPUProbe=cpu_probe
memVals.1=ram_total
xMonitorPath=/config/xos-monitoring/probes
memVals.0=ram_free
monitorType=ganglia
cpuVals.0=cpu_usage
xMonValName=value
```

Now you need to generate your client configuration files. You could run xconsole_dixi or xsub to generate them:

```
$ xconsole_dixi
Ctrl-C
```

Now modify them:

```
$ emacs $HOME/.xos/XATIConfig.conf

useSSL=false
certificateLocation=/etc/xos/truststore/certs/xati_dummy.pem
privateKeyLocation=/etc/xos/truststore/private/xati_dummy.pem
trustStoreSSL=/etc/xos/truststore/certs/dixi_ssl/

cdaCertificatePath=/etc/xos/truststore/certs/corehost-cda.crt
clientKeyPath=$YOUR_HOME/.xos/truststore/private/user.key
userCertificateFile=$YOUR_HOME/.xos/truststore/certs/user.crt
clientCertificatePath=$YOUR_HOME/.xos/truststore/certs/user.crt

cdaaddress.port=65000
xosdaddress.port=60000
address.port=10000

cdaaddress.host=131.254.201.16
address.host=131.254.201.20
xosdaddress.host=131.254.201.20
xosdaddress.externalAddress=131.254.201.20

Quit emacs.
```

5.3.8 Mount your XtreemFS Volume

XtreemFS permits you to save your data and binaries.

To see your own volume :

```
$ xtfs_lsvol http://xfs_core_ip:32636/
user-c4b32574-cf06-47e7-b960-97e7f6b994a4 -> 00065BBD8E7C900B51481CF8
```

Do you recognize you ID number ? It is user-c4b32574-cf06-47e7-b960-97e7f6b994a4. You need it to mount your XFS volume. Create a folder and mount the volume:

```
$ mkdir $HOME/MyVolume
Root # xtfs_mount -o dirservice=http://xfs_core_ip, \
       volume_url=http://xfs_core_ip:32636/user-c4b32...994a4, \
       direct_io,allow_other $HOME/MyVolume
```

allow_other permits to access the data without being root.

Add a job file in your volume:

```
No root $ emacs $HOME/MyVolume/cal.jsdl

<?xml version="1.0" encoding="UTF-8"?>
<JobDefinition xmlns:jsdl="http://schemas.ggf.org/jsdl/2005/11/jsdl">
 <JobDescription>
  <JobIdentification>
   <Description>Execution of cal</Description>
   <JobProject>Test</JobProject>
  </JobIdentification>
   <Application>
    <POSIXApplication xmlns:ns1="http://schemas.ggf.org/jsdl/2005/11/jsdl-pos
     <Executable>/usr/bin/cal</Executable>
    </POSIXApplication>
   </Application>
 </JobDescription>
</JobDefinition>

Quit emacs.
```

To umount this volume:

```
Root # umount $HOME/MyVolume
$ ls $HOME/MyVolume
(nothing)
```

Mount it again:

```
Root # xtfs_mount -o \
       volume_url=http://xfs_core_ip:32636/user-c4b32...994a4, \
       direct_io,allow_other $HOME/MyVolume

$ ls $HOME/MyVolume
cal.jsdl
```

We will use it for running the job.

5.3.9 Run a job with the AEM

(for killing the XOSd if necessary)

```
Root # service xosd stop
Ctrl+C it once the XOSd is finished.
```

XOSd must be running on 131.254.201.16 Core node and 131.254.201.21 Resource node. Link your Client XOSd to Core XOSd:

```
Root # service xosd start
```

Check there is a registered resource :

```
$ rca_list_registered
```

To run the job :

```
$ xsub -f $HOME/MyVolume/cal.jsdl
```

In one resource XOSd console you have :

```
September 2008
Su Mo Tu We Th Fr Sa
    1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
```

It's over. Now you can run more jobs described in the previous section or umount your XtreemFS volume:

```
Root # umount $HOME/MyVolume
```

6 Installing and configuring XtreemOS

6.1 Installing and Configuring the XtreemOS Root Certification Authority (Root CA)

The Root CA is the top level of the trust mechanism in XtreemOS. It is a critical part in the XtreemOS Public Key Infrastructure (PKI). To achieve and maintain the level of trust required by users of an XtreemOS Grid, the Root CA must be operated only on one machine. This host must be a physically-secure core node to avoid compromise of the Root CA private key, which would destroy any trust placed on the Root CA. Some organisations may choose to run the Root CA on a machine which isn't connected to a network, to eliminate any risk of intrusion.

The Root CA comprises root entity credentials which are trusted by all participants in an XtreemOS Grid, and a mechanism to create application certificates that identify other XtreemOS core services.

The package `rootca-config` contains the configuration files for creating a Root CA, and for creating application certificates from Certificate Signing Requests.

Install the `rootca-config package`. This places configuration files in `/etc/xos/config/openssl`. Decide on a directory to hold the files related to the Root CA, for example, `/opt/xtreemosca`. Then run the following command:

```
$ init-rootca /opt/xtreemosca
```

Figure 2: Creating the directory structure for the XtreemOS Root CA.

This step only needs to be carried out once.

The next step is to create the Root CA private key and self-signed public key certificate. This step is required when creating the Root CA, and when the Root CA certificate expires (with the default settings, every 365 days). The new Root CA certificate should be distributed before the current one expires.

The OpenSSL configuration is defined in the file `/etc/xos/config/openssl/create-rootca-creds.conf`. The section `[ root_ca_distinguished_name ]` can be modified to change the certificate fields `commonName`, `organizationName` and `organizationalUnitName` as required.

The command in figure 3 creates the root CA private key and public key certificate:

```
$ create-rootca-creds /opt/xtreemosca
```

Figure 3: Creating the Root CA private key and public key certificate.

You will be prompted for a passphrase - this protects the private key, and is required when using the Root CA to create application certificates from Certificate Signing Requests (CSRs). This passphrase must be kept secret to prevent use of the private key by anyone other than the operator of the Root CA.

The private key is created in the sub-directory `private` under the Root CA (in this case, `/opt/xtreemosca/private/xtreemos.key`). The public key certificate of the Root CA is the XtreemOS 'root certificate'. It is created in the sub-directory `public` of the Root CA directory (in this example, `/opt/xtreemosca/public/xtreemos.crt`). The XtreemOS root certificate needs to be installed on all other machines in this XtreemOS Grid. The certificate can be placed in `/etc/xos/truststore/certs/xtreemos.crt` on these machines.

The Root CA is now ready for its operational role. This consists of processing Certificate Signing Requests (CSRs) from administrators of core node (for applications such as CDA, RCA and VOPS servers, and XtreemFS client and servers). This is described in Section 7.1.2, 'Operating the Root Certificate Authority'.

6.2 Virtual Organization Management

6.2.1 Configuring X-VOMS

X-VOMS (XtreemOS Virtual Organization Management Service) is an advanced Virtual Organisation (VO) management service for supporting secure and flexible collaborations and resource sharing among people, projects and organisations. It is written in Java and back by a (Hibernate-based) X-VOMS database schema. Like other VO management software packages, X-VOMS provides a set of APIs for managing identity, attributes, and VO membership of users and resources.

X-VOMS can be used as a backend of different presentation frontends: a web application (allowing the access via a web browser), and a OS daemon service (allowing the access via a OS command line console, or directly from a user application). In *the current release*, X-VOMS is not a standalone service. It attaches to the VOLife web frontend to provide (part-of) its VO management capabilities to end users. In the future releases, the daemon frontend of X-VOMS will be offered so that applications can directly utilize X-VOMS functionalities.

X-VOMS manages, but does not distribute, credentials. It can be used with a Certification Authority (CA), such as the Credential Distribution Authority (CDA) service developed by the XtreemOS project, or a third-party attribute authority, to disseminate credentials.

X-VOMS also supports home volume creation for users of XtreemFS, a Grid file system being developed in the XtreemOS project.

This instruction assumes you know how to use MySQL (e.g. how to add a user in MySQL). For user management in MySQL, please read:

`http://dev.mysql.com/doc/refman/5.0/en/adding-users.html`

Your MySQL server must have a root password set. (This is not the default with some Linux distributions.) If there isn't a root password set, please follow the following instructions to set one (follow the steps describing the use of the `mysql` command; I couldn't get the `init-file` method to work):

`http://bit.ly/resetMySqlPassword`

Software prerequisites The current X-VOMS implementation relies on the following software:

- Hibernate 3.0[2]
- MySQL 5.1.6[3]

Major files and their location The steps needed to create the X-VOMS database and load it with data are encapsulated in the script xvoms_prepare_database.sh:

Configure the X-VOMS database:

```
Root # /usr/share/xvoms/bin/xvoms_prepare_database.sh
```

[2] http://www.hibernate.org/

[3] http://www.mysql.com

Running this script is sufficient to allow the following steps 'Installing the CDA' (6.2.2), 'Installing VOLife' (6.2.3) etc to be performed.

The following files described below are merely described for reference purposes.

The configuration files are located at: `/usr/share/xvoms/`. The X-VOMS library (xvoms-version.jar) is located at: `/usr/share/java/`.

- /usr/share/xvoms/scripts/setup.sql

  This script sets up the basic X-VOMS table schema. It is very important that you perform this step before starting to test any X-VOMS functionalities. Without setting up the tables, some security features (such as X-VOMS access control and authentication) cannot be demonstrated or your requests will be automatically denied.

  ```
  shell> mysql -u some_user --password=some_pass < setup.sql
  ```

  This line populates three tables: rules, actors, and actions, which are essential for X-VOMS.

- `/usr/share/xvoms/data/xvoms.txt`

  a sample xvoms database file, including both schema and some sample data (users, vos, vo attributes). To use this file to setup a sample xvoms database, perform the following steps:

  ```
  mysql> create database xvoms;
  ```

  ```
  mysql -u root --password=xxxxx xvoms < /usr/share/xvoms/data/xvoms.txt
  ```

  Two testing accounts, xtreemos-voadmin and xtreemos-vouser, are included in the sample xvoms.txt to allow testing X-VOMS. The password for both accounts is 'xtreemos'.

  To refresh the sample file, you can do:

  ```
  mysqldump -u root --password=xxxx xvoms -r /usr/share/xvoms/data/xvoms.t
  ```

- `/usr/share/xvoms/xsl/junit-noframes.xsl`

  xslt for transforming junit test reports (in XML) to html. This file should be used with the *source distribution* of X-VOMS, which contains build.xml to allow generation of junit test reports for X-VOMS functionalities.

- `/usr/share/xvoms/hibernate.cfg.xml` a hibernate configuration file for setting hibernate connection properties. The most notable settings are:

```
<property name="connection.url">jdbc:mysql://mysqlserver.someurl.com:3366/xvoms
<property name="connection.username">some_user</property>
<property name="connection.password">some_pass</property>
```

- /usr/share/xvoms/log4j.properties a log4j configuration file for setting hibernate logging properties. The most notable settings are:

  ```
  log4j.logger.org.hibernate=fatal
  log4j.logger.org.hibernate.SQL=fatal
  ```

- /usr/share/xvoms/MRC.properties a MRC/XtreemFS home volume configuration file for setting MRC server properties. The most notable settings are:

  ```
  mrc.host=testmrcserver.test.com
  mrc.port=32636
  ```

Other notes Apart from MySQL, you are free to choose any other JDBC-compliant database engines that Hibernate supports, including Oracle, DB2, Sybase, MS SQL Server, PostgreSQL, MySQL, HypersonicSQL, Mckoi SQL, SAP DB, Interbase, Pointbase, Progress, FrontBase, Ingres, Informix, and Firebird. See http://www.hibernate.org/344.html for more details.

6.2.2 Configuring and Running a Credential Distribution Authority (CDA) Server

This sub-section is for a Grid administrator running a CDA server.

The Credential Distribution Authority is implemented in the **cdaserver** package. For the first release of XtreemOS, the CDA server runs on only one core node in an XtreemOS Grid.

The standalone CDA client program can be used to obtain user VO credentials from the CDA, and is provided by the **cdaclient** package.

The CDA server issues XOS certificates to users. The server needs an application certificate issued by the Root CA to authenticate itself to the corresponding CDA client. This application certificate can be obtained by the procedure described in section 6.2.6. This procedure also produces a private key, which should be placed into /etc/xos/truststore/private/cda.key. The application certificate contains the service's public key, and should be placed in /etc/xos/truststore/certs/cda.crt.

As root, install the CDA server via:

```
Root # urpmi cdaserver
```

The following aspects of the CDA server are configurable by setting values in the file /etc/xos/config/cdaserver/cdaserver.properties:

- **cdaserver.keyFilename** — private key of CDA server - must be kept secure, readable only by owner.
- **cdaserver.keyPassphrase** — the private key is secured by a passphrase, the longer the better.
- **cdaserver.certFilename** — public key certificate of CDA server.
- **xtreemos.rootCertificate** — public key certificate of root CA.
- **cdaserver.sslAlgorithm** — cipher used by SSL.
- **cdaserver.sslHandshakeCipher** — the cipher used in initial SSL key exchange.
- **cdaserver.signatureAlgorithm** — algorithm used to sign the XOS-certificate returned to user.
- **cdaserver.validityDays** — number of days that certificate is valid for
- **cdaserver.validityHours** — number of hours that certificate is valid for
- **cdaserver.validityMinutes** — number of minutes that certificate is valid for

The validity of a certificate is calculated as (cdaserver.validityDays) days + (cdaserver.validityHours) hours + (cdaserver.validityMinutes) minutes. Any two of these values can be zero. Hence, the lifetime of certificates issued by the CDA server can be set on a fine basis, if required.

Other aspects of the CDA server operation are:

Connection to X-VOMS database - this is set in hibernate.cfg.xml

The level of logging, log file location, etc, are defined in log4j.properties.

Once configured, the server is started by issuing the following command:

```
Root # /sbin/service cdaserver start
```

The server writes its log files in `/var/log/cdaserver/cdaserver.log` by default.

6.2.3 Installing VOLife

Virtual Organization Lifecycle Management(VOLife) is a web-based tool for accessing various VO-related services in XtreemOS. Currently VOLife only supports the manuiplation of XVOMS database and the generation of XOS-Certs for users. Integration with runtime security services such as VOPS and RCA is still under development.

VOLife consists of two parts: backend and frontend. The backend is a light java wrapper around current security libraries. The frontend is a web application to be deployed into Tomcat. The backend provides a command-line utility which has almost the same functionality as the web front-end. But its main purpose is to test the integrity of data and the recommended way to use VOLife is via the web frontend.

Prerequisites The prerequistites of software packages include:

- MySQL 4.x or higher
- Tomcat 4.x or higher
- JRE 1.6 or higher

The web frontend of VOLife is written in JSP and to be deployed into Tomcat.

The running of VOLife also depends on the installation of XVOMS database. Please run `xvoms_prepare_database.sh` firstly to initialize the XVOMS database after you have installed the XtreemOS CD.

Installation To install VOLife, use the `urpmi` to install the following packages:

- **volife**. It contains a command-line script named voliferun.sh and several XVOMS database initialization files.
- **volife-backend**. The classes needed by command-line script mentioned above and the web frontend.
- **volife-frontend**. The web frontend application (i.e. a bundle of jsp, js and html files) to be deployed into Tomcat.

General configuration By default, VOLife is installed as a web application in the `webapps/` directory of Tomcat home directory (i.e. $CATALINA_HOME).

When generating XOS-Certs, VOLife uses settings in `/etc/xos/config/volife/volife.properties` to locate the CDA private key and certificate, and to supply the pass-phrase protecting the private key. These settings can be copied from the CDA configuration in `/etc/xos/config/cdaserver/cdaserver.properties`.

```
$ cat /etc/xos/config/volife/volife.properties
cdaserver.keyFilename=/etc/xos/truststore/private/cda.key
cdaserver.keyPassphrase=changeme
cdaserver.certFilename=/etc/xos/truststore/certs/cda.crt
```

The key pass-phrase should be changed to the actual pass-phrase protecting the private key.

The VOLife webapp puts the generated user private key and XOS-Cert files in the `certs/` directory. The path of the directory `certs/`) is relative to the directory from which Tomcat is launched. That means:

- For Tomcat, there should exist `certs/` directory under Tomcat `webapps/volifecycle` directory, and the user ID that Tomcat runs under (generally named `tomcat`) should have the write permission on the `certs/` directory.
- For the command line utility volife_run.sh, there should exist the `certs/` directory mentioned above.

To make sure your installation of VOLife works normally, check the directories as follows:

```
# cd /usr/share/tomcat5
```

```
# ls -l
```

There should have some lines indicating the certs/ directory exists and has the right permissions:

```
drwxrwxr-x 2 tomcat tomcat 4096 2008-07-10 13:09 certs/
```

Configuring home volume creation As part of its user creation process, VOLife contacts a MRC server (of XtreemFS) to create a home volume for users. VOLife looks for a file, called MRC.properties in *its classpath*[4] for MRC related settings (change these to fit your local settings):

```
mrc.host=mrchost.testcom.com
```

```
mrc.port=32636
```

For running smoothly, administrator has to guarantee dependent services have been started before VOLife service begins to work. The following lines are to check the services' status:

- Check if the mysql service is running:
  - `/etc/init.d/mysqld-max status`
- Check if the Tomcat service is running:
  - `/etc/init.d/tomcat5 status`

Consolidating Security of VOLife Server At default, encrypted connection is not opened in tomcat. This may sping out a security issue. users might be tricked to present their username/password to hijacked VOLife server.

Currently, the VOLife server is built on Tomcat in XtreemOS, we can privode an SSL connection between web browser and VOlife webapp by authenticating the server.

(1) Prepare a local certificate keystore. The keystore file is used in Tomcat to store certificate. In fresh tomcat, the keystore file can be created by following command:

```
$JAVA_HOME/bin/keytool -genkey -alias tomcat -keyalg RSA
  -keystore /path/to/my/keystore
```

You can specify its access pasword in this step. The defaut keystore file is located in user's home directory, named `.keystore` and given `changit` as password.

[4] Currently, this file locates at `webapps/volifecycle/WEB-INF/classes`.

(2) Create a Certificate Signing Request (CSR) for VOLife server. The certificate request should be signed by XtreemOS root certificate. By following command:

```
$JAVA_HOME/bin/keytool -certreq -keyalg RSA -alias tomcat
-file volife_server_certreq.csr
-keystore /path/to/my/keystore
```

If a custom keystore file has been specified in (1), it should be presented explicitly here.

(3) Send the CSR file to the operator of the XtreemOS root Certification Authority. Currently, the request file for the server has to be signed manually (Sending to administrator of XtreemOS root certificate by email).

(4) Import the server certificate and root certificate into local keystore. After signed certificate for VOLife server returned, it can be imported into Tomcat keystore file. Also, XtreemOS root certificate need to be import for authentication of chain certificate. The following instructions can help import both certifcates:

```
$JAVA_HOME/bin/keytool -import -alias root
  -keystore /path/to/my/keystore
  -trustcacerts -file /path/to/xtreemos_root_cert

$JAVA_HOME/bin/keytool -import -alias tomcat
  -keystore /path/to/my/keystore
  -trustcacerts -file /path/to/volife_server_cert
```

If above commands throw a error message:

```
keytool error:  java.security.cert.CertificateParsingException:
invalid DER-encoded certificate data
```

the certificates have to be convert to DER format first and then reimport in local keystore, because JAVA keytool only use DER format to keystore. The command:

```
openssl x509 -in volife_server_cert -inform PEM
-out volife_server_cert.der -outform DER
```

help to convert the PEM format to DER format.

(5) Edit the Tomcat configuration file to open secure socket. Tomcat need to be edited its `<Connector>` in the `$CATALINA_HOME/conf/server.xml` file, where `$CATALINA_HOME` represents the directory into which you installed Tomcat. Find and comment the non-SSL section:

```
<!-- Define a non-SSL HTTP/1.1 Connector on port 8080 -->
<Connector port="8080" maxHttpHeaderSize="8192"
       maxThreads="150" minSpareThreads="25"
       maxSpareThreads="75" enableLookups="false"
       redirectPort="8443" acceptCount="100"
       connectionTimeout="20000"
       disableUploadTimeout="true" />
```

and then, open the SSL-support section and edit the parameters as follows:

```
<!-- Define a SSL HTTP/1.1 Connector on port 8443 -->
<Connector protocol="HTTP/1.1"
       port="8443" maxHttpHeaderSize="8192"
       maxThreads="150" minSpareThreads="25"
       maxSpareThreads="75" enableLookups="false"
       disableUploadTimeout="true" acceptCount="100"
       scheme="https" secure="true" SSLEnabled="true"
       keystorePass="keystorepassword"
       keystoreFile="/path/to/my/keystore"
       debug="1" >
       <Factory clientAuth="false" protocol="TLS" />
</Connector>
```

- Restart Tomcat and test the webapp in browser. This final step is to test whether the SSL socket is in function. Restart your Tomcat service and try:

  ```
  https://localhost:8443
  ```

  If this works, it should prompt you to choose whether you accept certificate.

  I need to provide instructions in the 'Using VOLife' section on how users can import the XtreemOS root certifcate into their browser, to allow the SSL connection to the VOLife webapp to be trusted. Ian 24/10/08.

FAQ (1) Since the IcedTea JVM are not completely same as Sun JVM, the difference may cause the malfunction of SSL socket. The symptoms is like this:

```
...
Using CATALINA_HOME:   /usr/share/tomcat5
Using CATALINA_TMPDIR: /usr/share/tomcat5/temp
Using JRE_HOME:
- Error initializing endpoint
java.io.IOException: Invalid keystore format
        at sun.security.provider.JavaKeyStore.
         engineLoad(JavaKeyStore.java:650)
        at sun.security.provider.JavaKeyStore$JKS.
         engineLoad(JavaKeyStore.java:55)
        at java.security.KeyStore.load(KeyStore.java:1201)
        at org.apache.tomcat.util.net.jsse.JSSESocketFactory.
         getStore(JSSESocketFactory.java:287)
        at org.apache.tomcat.util.net.jsse.JSSESocketFactory.
         getTrustStore(JSSESocketFactory.java:261)
...
```

One of the solution is replacing the IcedTea JVM with Sun JVM.

To replace the IcedTea JVM with Sun JVM, you have to first download and install the Sun's JDK (recommended JDK1.6). And then `JAVA_HOME` and `JRE_HOME` have to be speficied to Sun's JVM.

- Edit `.bashrc` to add the new path of `JAVA_HOME` and `JRE_HOME` as global directory. Also, add the `$JAVA_HOME/bin` to global path.

  ```
  ...
  export JAVA_HOME=/path/to/jdk1.6.0_06/
  export JRE_HOME=/path/to/jdk1.6.0_06
  PATH=$JAVA_HOME/bin:$PATH
  ...
  ```

- Edit `/usr/bin/dtomcat5` and add the following path at file header.

  ```
  ...
  # OS specific support.  $var _must_ be set to either true or false.
  JAVA_HOME=/path/to/jdk1.6.0_06
  ```

```
    JRE_HOME=/path/to/jdk1.6.0_06

    cygwin=false
    ...
```

Finally, restart your Tomcat and try again.

(2) When SSL socket is enabled in VOLife under SUN JDK environment, another problem occurred due to the incompatibility of some IcedTea jar packages. A symptom is VOLife can not download user's certificate. And, in Tomcat log file, you can find the following exception message:

```
java.io.IOException: problem creating RSA private key:
java.io.IOException:
exception using cipher: java.lang.SecurityException:
JCE cannot authenticate the provider BC
...
```

This results from the IcedTea's encryption libraries are not compatible with Sun's. To solve the trouble, administrator has to replace the following IcedTea's jar packages: /texttt/usr/share/java/bcprov-1.39.jar with Sun's jar packages: `bcprov-jdk16-139.jar`
downloaded from:http://www.bouncycastle.org/.

6.2.4 Installing DIXI

DIXI (DIstributed Xtreemos Infrastructure) is a framework for running several of the VOM-related services. Before installing and using services for RCA and VOPS, it is essential to first install DIXI. DIXI is not required for CDA, X-VOMS and VOLife.

To install DIXI, use the `urpmi` to install the following packages:

- **dixi-main**. The package contains the core classes needed to run DIXI, XATI and the services.
- **dixi-services**. The classes implementing the main services, needed by other services.

Install XATI from the following package:

- **dixi-xati**. XATI, the collection of API and client-side access points for running clients. It also contains sample client programs and scripts for running them.

As a part of the package there is a script named **xosd** used for running the DIXI. The script runs an instance of the XtreemOS DIXI daemon which hosts services, configured in the configuration file (**XOSdConfig.conf** by default), provides the means for the services to communicate, and bridges services on different nodes.

The configuration files for **xosd** and all the services run by the daemon are placed in **/etc/xos/config/** by default.

The **xosd** takes the following command-line parameters:

- **-C** *config_path* sets the folder to contain all the configuration files used by the services. If the option is omitted, then the **/etc/xos/config/** folder is used. Overrides the **-c** directive for the xosd's configuration.
- **-c** *config_path* sets the path and file name to be used for DIXI daemon configuration. If the option is omitted, **/etc/xos/config/XOSdConfig.conf** is used instead. *Please note that if the value denotes a relative path, the daemon and the services will use config_path as the absolute path prefix.*
- **-r** *dir_path* instructs the daemon to set the DIXI root folder to the value of *dir_path*. By default this is the install path **/usr/share/dixi**.
- **-s** *server_ip[:port_num]* sets the ip address *server_ip* and, optionally, the port to *port_num*, of the *root xosd*. This directive overrides the related settings in the configuration files (*first release only*).

The *root xosd* is, in the first release, the daemon which keeps a directory of all the other xosd daemons and lists their services.

User can use **/etc/init.d/xosd** script for running, stopping and restarting **xosd**. Commands *start*, *stop* and *restart* are supported at this time, others will be supported soon:

```
$ sudo /etc/init.d/xosd {stop|stop|restart|condrestart|status}
```

The configuration file (**XOSdConfig.conf** by default) contains the following settings:

- **xosdRootDir** — the string containing the path to the DIXI daemon root.
- **useSSL** — indicates whether the communication should use SSL for security. Default value: *false*.
- **trustStoreSSL** — indicates the path to the folder containing the signer certificates trusted in the SSL handshakes.
- **trustStore** — indicates the path to the folder containing the signer certificates trusted in the AEM or other services.
- **certificateLocation** — the path to the public certificate used for the SSL handshakes. The certificate needs to be able to handle both server and client connections.
- **privateKeyLocation** — the path to the private key used for the SSL handshakes and for encrypting the communication.
- **networkInterface** — an optional option that tells the DIXI daemon which network interface to use for the inbound traffic, e.g., `eth0` or `eth1`. If **externalAddress**, **rootaddress.externalAddress** and **rootaddress.host** are left out from configuration (are commented out), **networkInterface** is used to set up IP proprely assuming XOSd root is running on this node.
- **xosdport** — the port the DIXI daemon will use for lisen to the inbound traffic.
- **externalAddress** — the IP address (in the format XXX.XXX.XXX.XXX) that nodes from other subnets can use to connect to this node. If this node is running behind a firewall, then the router or firewall that responds on the external address IP needs to forward the inbound traffic

arriving to port **xosdport** to this node. If the local subnet has multiple DIXI nodes, then one of the nodes will act as proxy, and the the firewall needs to forward the traffic on the port number as configured in that node's **XOSdConfig.conf**'s **xosdport** value.

- **xmlport** — the port used for the inbound XML connections from C applications using XATICA C library.

- **rootaddress.host** — the IP address (in the format XXX.XXX.XXX.XXX) of the **root DIXI daemon**. Can be the node's own address for a standalone or a root DIXI daemon. This address needs to be the one the root DIXI daemon's host's local address, as seen from the peers in the host's subnet. If the host is running on a different subnet than this node, the rootaddress.externalAddress has to be set to the address exposed to the current node.

- **rootaddress.port** — the port number the **root DIXI daemon** is listening to for requests. Default value is 60000.

- **rootaddress.externalAddress** — the IP address (in the format XXX.XXX.XXX.XXX) of the node visible from external network nodes. The node with this IP needs to have the port **rootaddress.port** forwarded to one of the nodes that on the internal network run the DIXI daemon.

- **services.size** — the number of services to be run by this DIXI daemon.

- **services.**X — one or more entries listing the services to be run by this DIXI daemon, with $0 \leq X <$ **services.size**. Make sure each service has its own X index, otherwise only the last entry with the same number will be used. There will be no errors if any of the indices are missing.

Figure 4 shows an example **XOSdConfig.conf** which configures the local DIXI daemon to connect to a *root daemon* running at the XtreemOS testbed node located at XLAB, using the SSL credentials of the xtreemuser01. The daemon will start up and serve the RCA client, AEM's Resource Monitor and Execution Manager, and a common Daemon and XML Extractor service.

The services and the DIXI daemons advertise their location as a combination of the local host's address, the port the service or the daemon is listening to, and the external address. The daemon that uses the configuration in Figure 4 would query the `eth0` network interface (the **networkInterface** option) and

```
# This is a sample configuration.
# The lines starting with the hash # are ignored.

# Security and SSL-related settings
trustStore=/etc/xos/truststore/certs/aem_trusted/
useSSL=true
trustStoreSSL=/etc/xos/truststore/certs/dixi_ssl/
certificateLocation=/etc/xos/truststore/certs/xosd_dummy.pem
privateKeyLocation=/etc/xos/truststore/private/xosd_dummy.pem

# Node’s parameters and root addresses
xosdRootDir=.
networkInterface=eth0
xosdport=60000
xmlport=55000
externalAddress=194.249.173.87
# This node connects to a testbed node at XLAB
rootaddress.host=194.249.173.88
rootaddress.externalAddress=194.249.173.88
rootaddress.port=60000

# Enumerate the services to be run on the node
services.size=7
# This service needs to be included for
# proper operation of xosd
services.0=eu.xtreemos.xosd.daemon.DaemonGlobal
services.1=eu.xtreemos.xosd.security.rca.client.service.RCAClientHandler
services.3=eu.xtreemos.xosd.resourcemonitor.service.ResourceMonitorHandler
services.4=eu.xtreemos.xosd.execMng.service.ExecMngHandler
services.6=eu.xtreemos.xosd.xmlextractor.service.XMLExtractorHandler
```

Figure 4: A sample DIXI daemon configuration file.

obtain its host’s local address (e.g., 192.168.0.30). It would combine this address with the information from **xosdport** and **externalAddress** to compose its communication address `///192.168.0.30:60000(/194.249.173.87)`, which also doubles as the node’s ID.

Connecting DIXI daemons from multiple nodes. In the first release, the DIXI daemons need to be connected explicitely to a central daemon.

This is obviously an unscalable solution, therefore in the future releases the interconnections will occur automatically via ADS. Currently, however, there should be a designated root node for the other nodes to connect to. In the test phase, this can be 194.249.173.88 at port 60000, or another designated node within your LAN.

To connect the xosd to another xosd, use either the **-s** command-line directive, or the **rootaddress.host** and **rootaddress.port** settings in the **XOSdConfig.conf**.

Secure communication (SSL). The communication between services on different nodes uses SSL by default. This can be turned off using the **useSSL** setting in the **XOSdConfig.conf**. **Note, however, that the nodes with SSL enabled will refuse the inbound plain-text connections.** For using SSL, we need to configure which private key and which certificate to use in the handshakes, as well as which certificates to trust. These settings can be set via **privateKeyLocation**, **certificateLocation** and **trustStoreSSL** settings in **XOSdConfig.conf**, respectively. The certificates valid for the handshakes and accepted by other xosd nodes by default are the ones issued by the **CDA** (user's credentials) and the ones issued by the **RCA** (machine credentials). When running the DIXI daemon for the first time, we recommend first obtaining a dummy certificate manually created by the RCA admin, copying it to the truststore/XATI folder and pointing **XOSdConfig.conf** settings to the key and certificate files. The folder pointed to by **trustStoreSSL** should contain the public certificates of the certificate signing authorities (CA certificates) that we are trusting to connect to our node.

Traversing NAT. The DIXI framework supports connecting and communicating with nodes that are on another subnet and behind the firewall. However, this only works if the target node's subnet contains a node that runs xosd, is visible from external networks by port-forwarding, and the target node's firewall allows inbound and outbound traffic to the visible node on port designated for the xosd communication (60000 by default). For proper operation, every xosd needs to have **XOSdConfig.conf** configured so that **externalAddress** contains the externally visible address that maps into the local LAN address space and which runs its own xosd.

Running xosd as root. In principle, an ordinary user can run xosd. However, many of the services are VO-critical, they run on a secure node or access local resources. In these cases, the **xosd** needs to be run with sudo or from root account. For running xosd use **/etc/init.d/xosd** script with command *start* or *restart*.

Stopping xosd. The **xosd** can be stopped by sending the break (Ctrl+C) signal to the process, or using **xosdkill** XATI script. Prefered way is to use **/etc/init.d/xosd** script with command *stop* or just executing following command:

```
[root@localhost ~]# service xosd stop
```

DIXI logging Property file **/usr/share/dixi/log4j.properties** assigns path to target file for logging facility used in DIXI. By default it points to **/var/log/xosd/xosd.log** but someone can easily change property file to point to different location. There is an issue when running XATI (described below) in user mode without superuser rights. Since XATI uses the same logging facility and the same property file, there can be an error message saying that user does not have rights to append to target file. By changing the line in **log4j.properties**

```
...
log4j.appender.A3.File=/var/log/xosd/xosd.log
...
```

this permission issue can be easily resolved.

XATI. The client programs use XATI to access the functionality of the services. XATI needs a running DIXI daemon to connect to. This can be a daemon running on any node accessible from the client node. However, it is preferable to run an instance on the same node that runs the XATI program. One possible issue of connecting to a remote daemon is that the daemon can open multiple ports towards the client, while two daemons use a single channel to communicate.

The XATI in all client programs use **XATIConfig.conf** file to read the settings from. By default, the configuration file is located in the **/home/username/.xos/** folder. The settings are as follows:

- **address.port** — the port number used by XATI to communicate with xosd.

- **networkInterface** — an optional option that tells the XATI which network interface to use for the inbound traffic, e.g., `eth0` or `eth1`.

- **xosdaddress.host** — the address of the DIXI daemon (xosd) this XATI program is connecting to.

- **xosdaddress.port** — the port number used by the DIXI daemon (xosd) this XATI program is connecting to.

- **xosdaddress.externalAddress** — the externally visible address of the xosd this XATI program is connecting to.

- **userCertificateFile** — the path and filename of the certificate issued by the CDA to the user. Usually, the user certificates are placed in **/home/username/.xos/truststore/certs/**.

- **useSSL** — indicates whether the communication should use SSL for security. Default value: *false*.

- **trustStoreSSL** — indicates the path to the folder containing the signer certificates trusted in the SSL handshakes.

- **certificateLocation** — the path to the public certificate used for the SSL handshakes. The certificate needs to be able to handle both server and client connections.

- **privateKeyLocation** — the path to the private key used for the SSL handshakes and for encrypting the communication.

- **userKeyFile** — this entry points to the path of the user private key file

- **address.host** — the address of the XATI daemon

Figure 5 shows a sample configuration file for client programs using XATI.

6.2.5 RCA

The Resource Certification Authority services run as DIXI services. Before installing it, please first install DIXI (Section 6.2.4).

RCA comes in two packages:

- **vom-rca-node** — This package contains the node level service which should run on each node capable of executing jobs.

```
# This is a sample XATIConfig.conf.

# Use this port for sending XATI requests
address.port=10000

# xosd details
xosdaddress.externalAddress=192.168.0.178
xosdaddress.host=192.168.0.178
xosdaddress.port=60000

# SSL
useSSL=false
trustStoreSSL=/etc/xos/truststore/certs/dixi_ssl/
certificateLocation=/etc/xos/truststore/certs/xosd_dummy.pem
privateKeyLocation=/etc/xos/truststore/private/xosd_dummy.pem

# Security and credentials
userCertificateFile=/home/matej/.xos/truststore/certs/user.crt
userKeyFile=/home/matej/.xos/truststore/private/user.key

address.host=192.168.0.178
```

Figure 5: A sample XATI configuration file.

- **vom-rca-server** — This package contains the core-side service which usually runs on one node within a physical organisation.

To install the necessary software, simply use `urpmi` with the name of the package. In order to actually run either RCA client or the RCA server, the DIXI daemon's configuration file (**XOSdConfig.conf** by default, please refer to Section 6.2.4 for more details) needs to have its handler enabled. The configuration files used by the RCA are placed into **/etc/xos/config/** by default.

For the normal operation of the RCA client, the node running the RCA client's service also needs to run the AEM's Resource Monitor.

Enabling services in DIXI daemon's configuration. The **XOSdConfig.conf** file needs to contain one **services.***X* line for each service handler that needs to be run on the node. The possible handlers for the RCA services are as follows:

- **RCA server**: eu.xtreemos.xosd.security.rca.server.service.RCAServerHandler
- **RCA client**: eu.xtreemos.xosd.security.rca.client.service.RCAClientHandler

Configuring core-level RCA service. The RCA server service creates and uses the **RCAServerConfig.conf** to obtain the configuration:

- **certDNLocation** — the location of the organisation covered by the RCA server. The value is a part of the distinguished name (DN) of a certified resource.
- **certDNCountry** — the country of the organisation covered by the RCA server. The value is a part of the distinguished name (DN) of a certified resource.
- **certDNOrganisation** — the name of the organisation covered by the RCA server. The value is a part of the distinguished name (DN) of a certified resource.
- **certDNOrganisationUnit** — the name of the organisation unit covered by the RCA server. The value is a part of the distinguished name (DN) of a certified resource.
- **daysCertValidity** — The number of days the certificate will be valid, starting from the day of certification and expiring this number of days later.
- **privateKey** — The path to the server's certificate authority's private key.
- **certificateFileName** — The path to the server's certificate authority's public key/certificate.
- **cdaPassword** — The server's certificate authority's public key's passphrase.
- **keyPassword** — The server's certificate authority's private key's passphrase.
- **rcaDBFile** — The path to the file containing the RCA DB.
- **attributeType** — the type of the attribute certificates. Use V2 for attribute certificates, or V3 for certificates with attributes stored in extensions. The default value is V3, and it is a recommended value for compatiblity with openssl libraries.

The RCA server requires a private key and a certificate signed by a certification authority that is trusted by the nodes in the XtreemOS. The RCA server will use the certificate signed by a commonly trusted root authority to sign the machine certificate requests. The steps for creating the certificate for the RCA are similar to those described in Section 6.2.2 for the CDA server. The location of the private key and the certificate are defined by **privateKey** and **certificateFileName** of the **RCAServerConfig.conf**, respectively.

Configuring node-level RCA service. The RCA client service creates and uses the **RCAClientConfig.conf** to obtain the configuration:

- **cdaCertificateFileName** — the path to the RCA server's certificate authority's public key/certificate.
- **resPrivateKeyFileName** — the path to the resource's private key.
- **resIdentityCertFileName** — the path to the resource's identity certificate (public key).
- **resAttributeCertFileName** — the path to the resource's attribute certificate (attribute certificate).
- **resAttributeCertExtFileName** — the path to the resource's attribute certificate (attributes stored in an extension).
- **resVOAttributeCertIncoming** — the path to the folder that will store the attribute certificates pushed from the RCA Server.

6.2.6 Preparing Core Services - CDA, RCA, and VOPS servers, XtreemFS servers and XtreemFS mount client

Generic instructions for preparing core services The XtreemOS core services require configuring with an application certificate before they can be started. In addition, services using the DIXI message bus require configuring of the DIXI and XATI subsystems (see sections 6.2.4).

Most application certificates are used to authenticate core services to client programs. For mounting XtreemFS filesystems, one mode of use is to use the application certificate for the `xtfs_mount` application to authenticate the client host to the XtreemFS server. Alternatively, the XtreemFS mount client can also use an XOS-certificate if the client is being run on behalf of a single user.

Prerequisite for installing any core service application. The following conditions apply:
Before installing any server, the XtreemOS Root Certificate Authority must be active in your XtreemOS Grid. See Section 6.1 for details.
The `create-csr` package must be installed; it contains the **create-csr** command and an OpenSSL configuration file to create a certificate signing request (CSR) file for an application. This CSR is then sent to the operator of the Root CA to obtain the application certificate.

The steps involved are shown below.

The **create-csr** command creates a Certificate Signing Request (CSR) file. The arguments to this command are:

- the host name — this is encoded in the subjectAltName extension field of the certificate, and as part of the Subject CN field. The Fully-Qualified Domain Name for the host is required, not its IP address. Some client programs, such as the CDA client, will check this field during the SSL handshake against the FQDN of the server they are attempting to connect to.
- the name of your organisation.
- the name of the application. This is incorporate into the Subject CN field as <fqhn>/<application>. E.g. for a CDA server at host.org.domain, the Subject field would include `CN=host.org.domain/cda`.
  Legitimate values for the application argument are:
  - `cda` The Credential Distribution Authority server
  - `vops` The VO Policy Service server
  - `mrc` The XtreemFS Metadata and Replica Catalogue Server
  - `dir` The XtreemFS Directory Service
  - `osd` The XtreemFS Object Storage Device server
  - `xtfs_mount` The XtreemFS mount client

An example, creating a request for a application certificate, where `host.org.domain` is replaced with either the Fully-Qualified Domain Name for the host, or its IP address. The last argument to this command identifies the type of service/client that this certificate will be used by.

```
create-csr host.org.domain "My Organization" cda
```

Figure 6: Creating a request for a CDA application certificate.

This command produces a private key for the application in `host.org.domain-cda.key`, and a CSR in `host.org.domain-cda.csr`. Send this CSR file to the administrator of the Root CA in your organization to get an application certificate (e.g. `host.org.domain-cda.crt`) in return. Install this application certificate in `/etc/xos/truststore/certs/cda.crt` and the private key in `/etc/xos/truststore/private/cda.key`.

The passphrase protecting the key can be specified in the properties/configuration file of the server it is to be used with. In this case, you must ensure that the file containing the passphrase is only readable by the owner of the service itself, e.g. for the CDA server, the properties file should only be readable by ’`cdauser`’.

Connecting the CDA server to the X-VOMS database The CDA server uses the Hibernate ORM library to retrieve VO attributes from the X-VOMS. Hibernate uses a JDBC connection that is specified by the parameters in the Hibernate configuration file, `hibernate.cfg.xml`. The settings that may need to be changed here are `connection.username` and `connection.password`.

6.2.7 VOPS

VOPS is a **core-level service** which, due to usage of the DIXI framework, runs as a service using DIXI communication stages. Please refer to Section 6.2.4 for details. VOPS has to be started in a way like other XOS daemons are: using *xosd* script provided in a bundle containing VOPS package. First, administrator has to set up **XOSdConfig.conf** and **VOPSConfig.conf** appropriately. **ResMng.conf** (on server, where ResMng service is running) has to be configured appropriately to use VOPS, see also figure 9. VOPS is a server primarily intended serving requests and forwarding answers from/to resource discovery services and therefore it needs private key and public certificate to be able to digitally sign its decisions before forwarding them to services. Services querying VOPS should have access to VOPS public certificate to be able to check authenticity of its answers. To obtain VOPS server key/certificate please refer to section 6.2.2 where steps for obtaining server certificate is described.

To be able to run VOPS server using DIXI framework, XOSdConfig.conf has to include a line under services array (see figure 7).

```
# e.g. add a service handler under 8-th entry
services.7=eu.xtreemos.xosd.security.vops.service.VOPSHandler
```

Figure 7: A sample VOPS configuration file.

If VOPSConfig.conf does not exist yet, you can run xosd and stop it. This way VOPSConfig.conf is automatically generated under /etc/xos/config, where you can edit it manually (see figure 8).

```
globalVOPS.host=194.249.173.88
globalVOPS.port=60200

enableAccessControl=true

VOAdminRoles.size=15
VOAdminRoles.0=role_get_VOAttributes4

ResourceAdminRoles.size=15
ResourceAdminRoles.0=res_role_get_VOAttributes

serviceKey=/etc/xos/truststore/private/vopsserver.pem
policyStorage=/usr/share/dixi/VOPS/files/policy/testStorage
```

Figure 8: A sample VOPS configuration file.

- **globalVOPS.port** and **globalVOPS.host** are not used yet, so you can comment these two lines or just simply ignore them.
- **enableAccessControl** enables or disables access control: if enabled, extension (role) from user certificate is checked whether it is one from roles listed under **VOAdminRoles** or **ResourceAdminRoles**.
- **VOAdminRoles.size** is the size of array defining VO administrator roles.
- **VOAdminRoles** are roles of users which are permitted to manipulate with XACML policies. These roles must be same as roles specified in certificates (VO administrator roles).

- **ResourceAdminRoles.size** is the size of array defining resource administrator roles.

- **ResourceAdminRoles** are roles of users which are permitted to manipulate with XACML policies. These roles must be same as roles specified in certificates (resource administrator roles).

- **serviceKey** is VOPS's private key used to sign responses.

- entry **policyStorage** points to storage (XML files) which contains user policies and resource policies defining access control to users over these resources.

```
#Properties File for the client application
#Thu Jun 26 13:08:14 CEST 2008
VOPSPubCert=/etc/xos/truststore/certs/vopsserver.pem
testVOPS=true
```

Figure 9: A sample ResMng configuration file.

While resource discovery services have to check authenticity of the VOPS's answers, the node running **ResMng** service has to include next lines in its configuration files. List of entries under **ResMng** configuration file:

- **VOPSPubCert** is path to public certificate of the vops server.

- **testVOPS** enables or disables calls to VOPS service.

It is important that if VOPS is to enforce policies over user queries, RCA client must run on resource node which is considered in query. VOPS needs to access RCA client service to obtain resource certificates from which attributes are considered in the query.

Packages:

- **vom-vops** —The VOPS service provides means to store and manage VO-level policies, to obtain the policy filters and the policy decisions on the VO level.

6.3 Application Execution Management

The AEM services mostly host in a DIXI framework. In order to install DIXI, please refer to Section 6.2.4. In order to run one or more AEM services, the DIXI daemon's configuration file (**XOSdConfig.conf** by default, please refer to Section 6.2.4 for more details) needs to have its handler enabled.

6.3.1 Core-level AEM services

The services and the software related to the core-level AEM can be found in the following package

- **aem-server**.

The package contains services which provide the job management, job directory and resource management.

Enabling services in DIXI daemon's configuration. The **XOSdConfig.conf** file needs to contain one **services.**X line for each service handler that needs to be run on the node. The possible handlers for the core AEM services are as follows:

- **Job Manager**: eu.xtreemos.xosd.jobmng.service.JobMngHandler
- **Job Directory**: eu.xtreemos.xosd.jobDirectory.service.JobDirectoryHandler
- **Resource Manager**: eu.xtreemos.xosd.resmng.service.ResMngHandler

Configuring Job Manager. JobMng service uses **JobMng.conf** to read and store its configuration. This configuration file was used in early development stage and at this point it is ignored by the JobMng service. This file consists of the following settings:

- **useVOPS**
- **VOPS.host**
- **VOPS.port**

Configuring Resource Manager. ResMng service uses **ResMng.conf** to read and store its configuration. This file consists of the following settings:

- **VOPSPubCert** — the path and the filename of the VOPS's public certificate used for checking the VOPS response's data signatures.
- **testVOPS** — experimental. By setting the value to false the resource manager does not check VO policies with VOPS. The default value is true.

6.3.2 Node-level AEM services

The services and the software related to the node-level AEM can be found in the following package

- **aem-node**.

To run the node-level AEM services which are responsible for providing the resource information of the local node, and executes jobs. They require that the local node's kernel is compiled with connectors enabled, and that the Ganglia monitoring is installed and running on the node. Further, it requires the XtreemOS PAM module support to be installed and configured, as well as the RCA client service (Section 6.2).

To enable the possiblity of job execution on the node, the node also needs to provide the details on the resource. The security services also depend on the node's ability to send its VO-related attribute certificate on request. Hence, the node running AEM's Execution Manager also needs to run AEM's Resource Monitor and VOM's RCA client.

Enabling kernel connectors. Kernel connectors need to be compiled into the kernel, as using the cn module is not enough. Mandriva install CD already includes compiled kernel connectors, so nothing is to be done. In the case you need to compile it manually, please refer to Annex I.

Enabling services in DIXI daemon's configuration. The **XOSdConfig.conf** file needs to contain one **services.**X line for each service handler that needs to be run on the node. The possible handlers for the node-level AEM services are as follows:

- **Resource Monitor**: eu.xtreemos.xosd.resourcemonitor.service.ResourceMonitorHandl

- **Execution Manager**: eu.xtreemos.xosd.execMng.service.ExecMngHandler

Configuring resource monitor. The ResourceMonitor service uses **ResourceMonitorConfig.conf** to read and store its configuration. The file contains the following settings:

- **monitorType** — the type of external monitor used. The default value *ganglia* sets the Resource Monitor to use Ganglia monitoring system. Alternative value, *xmonitor*, is not explained here.

- **gangliaPort** — the port number that the Ganglia monitoring listens at for the requests. Default port number for Ganglia monitoring is 8649.

- **cpuVals.size** — *xmonitor*-related setting.

- **memVals.size** — *xmonitor*-related setting.

- **cpuVals.0** — *xmonitor*-related setting.

- **memVals.1** — *xmonitor*-related setting.

- **memVals.0** — *xmonitor*-related setting.

- **xMonitorPath** — *xmonitor*-related setting.

- **xMonMemProbe** — *xmonitor*-related setting.

- **xMonCPUProbe** — *xmonitor*-related setting.

- **xMonValName** — *xmonitor*-related setting.

6.3.3 AEM clients

The clients to the AEM services use XATI to access to the services' functionality. please refer to Section 6.2.4 for more details on XATI.

6.4 Job execution preparation

In order for the jobs submitted to the node to be able to successfully run, the PAM extensions and the Account Mapping Service need to be properly configured first [4]. The following checklist covers the necessary steps, performed as a root user.

I think we can remove the need to create a separate directory for PAM to find certificates in. We should tell PAM to read certificates directly from /etc/xos/truststore/certs. Ian 28/10/08

1. Ensure xos_amsd is running.
   - The `ps -A | grep xos_amsd` command displays whether the daemon is currently running. If it is not, then
   - If this is before the first time running xos_amsd, it can be started by calling `xos_amsd -init`.
   - Otherwise it can be started using `/etc/init.d/xos-amsd start`.
2. Check the PAM extension configuration.
   - Create a folder to contain the trusted CA certificates for the PAM extensions, e.g., `/etc/xos/truststore/certs/pam` and populate it with the root CA certificate and the certificate(s) that sign trusted user certificates. The latter need to be named as their hash, and have to have the .0 extension.
     - `mkdir -p /etc/xos/truststore/certs/pam`
     - `ln -s /etc/xos/truststore/certs/xtreemos.crt \ /etc/xos/truststore/certs/pam/xtreemos.crt`
     - `export CDA_CERT=/etc/xos/truststore/certs/cda.crt`
     - `` ln -s $CDA_CERT /etc/xos/truststore/certs/pam/`openssl x509 -noout -hash -in $CDA_CERT`.0 ``
   - In a text editor, open `/etc/xos/nss_pam/pam_xos.conf`,
   - Set `VOCACertDir` to `/etc/xos/truststore/certs/pam/`
   - Set `VOCACertFile` to `xtreemos.crt`
3. Create the mappings for the VO.
   - To learn the VO's globally unique identifier, you can refer to the VOLife (log in, then navigate to Home / Join a VO, the value is the GVID column). You may find it easier examine the XOS-Certificate of a user that belongs to the VO:

        - Assuming the certificate is named `user.crt`, then issue `view-xos-cert user.crt`
        - In the output, the VO's global unique ID is printed after the label "GlobalPrimaryVOName:".
    - first create a mapping for the user by calling `xos-policy-admin-am -dn \* -vo $VO_UNIQUE_KEY -locnam \* -drvname root -drvparam 2510`
    - then create a mapping for the group by calling `xos-policy-admin-gm -grp \* -vo $VO_UNIQUE_KEY -locgrp \*`

4. Check the ability to execute jobs on the node. This involves using a user XOS-Certificate which includes the proper VO information:

    - `pam_app_conv -pem user.crt`
    - The test succeeds if it shows no "Ooops..." error and puts the prompt in the environment's inner shell.

5. Check or create PAM AEM configuration (/etc/pam.d/aem). Its content should be:

```
#%PAM-1.0
auth sufficient         /lib/security/pam_xos.so
account sufficient      /lib/security/pam_xos.so
session sufficient      /lib/security/pam_xos.so
```

6.5 SSL Configuration in AEM

Before reading this, see section 6.2.4, paragraph *Secure communication (SSL)*

When configuring SSL in DIXI, first certificates for initialising SSL context must be generated. After that certificates must be distributed some way (e.g. publishing them publicly). There are still some known issues in SSL configuration:

- if one instance of DIXI uses SSL, all instances must use SSL,
- to generate certificates for SSL context, one instance of AEM running RCA server 6.2.5 without SSL must be used and than restart it with SSL turned on and distribute generated certificates (public and private key!),
- client authentication is required by default,

These issues should be updated in next releases of DIXI framework. For SSL initialisation first run AEM with SSL turned off:

- edit **/etc/xos/config/XOSdConfig.conf** and set *useSSL* to *false*.
- add to **/etc/xos/config/XOSdConfig.conf** line *services.1=eu.xtreemos.xosd.security.rca.server.service.RCAServerHandler* if it is not already there.

Obtain RCA certificate using steps described in 6.2.6 or copy them from *https://scm.gforge.inria.fr/svn/xtreemos/WP3.3/trunk/Support/2008ReviewDemo/node0/* if you have access to Inria's GFORGE SVN (certs/rcaserver.pem and private/rcaserver.pem) and put them into **/etc/xos/truststore/certs** and **/etc/xos/truststore/private** respectively.

Before running *dixi_test* also edit **XATIConfig.conf** (e.g. /root/.xos/XATIConfig.conf or /home/user/.xos/XATIConfig.conf, depends on user executing dixi_test) and set *useSSL* to *false*.

Now restart XOSd and run *dixi_test -RCA dc* to generate dummy SSL certificates (*DummyResCert.pem* and *DummyResKey.pem*) which are placed under **/usr/share/dixi**.

```
$ sudo service xosd restart
$ dixi_test -RCA dc
Returned from service call: successMethod
Creating a dummy certificate.
$ ls /usr/share/dixi/
... DummyResCert.pem DummyResCert.pem ...
$ sudo service xosd stop
$ sudo ln -s /usr/share/dixi/DummyResCert.pem
/etc/xos/truststore/certs/xosd_dummy.pem
$ sudo ln -s /usr/share/dixi/DummyResKey.pem
/etc/xos/truststore/private/xosd_dummy.pem
$ sudo ln -s /usr/share/dixi/DummyResCert.pem
/etc/xos/truststore/certs/xati_dummy.pem
$ sudo ln -s /usr/share/dixi/DummyResKey.pem
/etc/xos/truststore/private/xati_dummy.pem
```

Next step is distributing newly generated certificates (*DummyResCert.pem* and *DummyResKey.pem*), putting them into **/use/share/dixi/** on each machine running XATI or XOSd and creating links in the same way as above:

```
$ ls /usr/share/dixi/
... DummyResCert.pem DummyResCert.pem ...
$ sudo service xosd stop
$ sudo ln -s /usr/share/dixi/DummyResCert.pem
/etc/xos/truststore/certs/xosd_dummy.pem
$ sudo ln -s /usr/share/dixi/DummyResKey.pem
/etc/xos/truststore/private/xosd_dummy.pem
$ sudo ln -s /usr/share/dixi/DummyResCert.pem
/etc/xos/truststore/certs/xati_dummy.pem
$ sudo ln -s /usr/share/dixi/DummyResKey.pem
/etc/xos/truststore/private/xati_dummy.pem
// Set useSSL to true:
$ sudo vim /etc/xos/config/XOSdConfig.conf
// Restart XOSd:
$ sudo service xosd restart
```

SSL context should now be initialized between multiple XOSds or XATI and XOSd using dummy certificates. All instances running XOSd or XATI have to set *useSSL* property inside XATI/XOSd configuration files to true.

6.6 ADS Bamboo – the DHT used by SRDS

The **ADS_Bamboo** Module provides DHT service to the SRDS module. This is the only DHT service provided by the SRDS in the first release of XtreemOS. It is functionally equivalent to the standard Bamboo DHT, with a few extensions regarding configurability of the time-out delays used in several points of its implementation[5] ADS_Bamboohas been modified to add additional configuration parameters, which are used to set the default timeouts for all DHT primitives.

The **ADS_Bamboo** overlay network places one Java process on each active XtreemOS node, in order to set up the DHT overlay. Exactly one of these nodes need to be configured as the bootstrap one. This means that the bootstrap node will open one additional TCP port, and it will be used as the initial contact point for all new nodes joining the Bamboo overlay. **ADS_Bamboo** code is run in its own separate JVM, that is started by a shell script.

[5] More specifically, several timeout in the original Bamboo Library are fixed to 5 seconds in the code, regardless of any option in the Bamboo configuration file.

Important notice Bamboo underwent no further development since mid 2006. Its use within XtreemOS is going to be discontinued, it will be replaced by another DHT library. Eventually, the ADS_Bamboomodule will disappear from the XtreemOS distribution.

6.6.1 Installing ADS_Bamboo

Installation of ADS_Bamboois performed using `urpmi`.

```
$ urpmi srds
```

The package contains ADS_Bamboo, and has among its dependencies on an additional package containing all jar files the original Bamboo depends upon. ADS_Bamboois a prerequisite of the **SRDS** package, the former is automatically installed if you install the latter.

6.6.2 Configuring ADS_Bamboo

The configuration for the ADS_BambooDHT is almost all in the file `/etc/xos/config/Bamboo/stdconf.cfg`, with a few details to be checked in the file `/etc/xos/config/Bamboo/run-java`

- **run-java**: This script file is used from within the SRDS code to lauch the Bamboo DHT. It is configurable with the proper directory where both native and jar libraries are located. Under XtreemOS, this directory should be `/usr/share/java`.
- **stdconf.cfg**: This file contains the actual Bamboo configuration. It is structured in sections.
  - The first section needing changes is the **global** subsection, the field **node_id** should take the IP:port of the current node. The IP must be in numeric fashion and public. The port number used by Bamboo is 3630.
  - In the **Router** section, the **gateway** line, we have the IP (public) and port of the bootstrap node of Bamboo. The gateway (bootstrap) IP must be the same for all nodes. Here too the port number should be 3630. (as we do not need more Bamboo nodes per machine in the first release of XtreemOS).

- The **Storage** section specifies the directory where Bamboo stores a local copy of DHT content. The same path should be specified in `stop_srds.sh` script in SRDS root project. This local copy is used to recover information if the node fails, and should be deleted when doing a clean shutdown.
- In the **Gateway** section at the line **port** we have to insert the port number of the gateway to use for ingoing calls; we use 3632. This is for puts and gets to the DHT, this is NOT the same of routing section; bad name choice but not ours.

  NOTE: The attribute drop prob in the Network section *MUST* be 0.0: this floating point field is used to simulate packet loss in Bamboo.
- The section **WebInterface** is used to bring up the web server for each node composing the Bamboo overlay network. Through the web server you can observe the leaf set, neighborhood set, the storage and other information about the node. To access the web interface, point a browser at the url `http://node_ip_address:3632`.

6.7 Resource Selection Service

The RSS (Resource selection service) is a Java service providing a dedicate overlay network (exploiting the Cyclon communication layer) to efficiently locate computing resources based on their static attributes (CPU, memory amount and so on).

RSS interacts with SRDS, the two modules running inside the same JVM. There is one instance of each per computing resource. Beside that, The RSS network needs one additional process to manage nodes who want to join the RSS overlay. This process (called Recorder) has to be active exactly on one node and to be known to all the others.

6.7.1 Install RSS

The RSS module is installed with `urpmi` using the following command:

```
urpmi rss
```

6.7.2 Configure RSS

The configuration file for RSS is located in `/etc/xos/config/Rss/config.conf`

- **network_interface**: public interface used by the RSS (e.g. eth1)
- **bootstrap_address**: the address of the RSS bootstrap node (the recorder)
- **local_port**: the port number where the local RSS implementation will listen to requests from other nodes
- **bootstrap_port**: the port number of the Recorder that will be the bootstrap node of Rss overlay

It is possible to configure the log output. To disable logging of output, comment the corresponding line.

In order to run the RSS recorder (tracker) on the machine at **bootstrap_address**, from that machine, go to the directory with XtreemOS jars (`/usr/share/java`) and run

```
java -cp DIXIMain.jar:srds.jar:xtreemrss.jar:log4j-1.2.14.jar \
eu.xtreemos.ads.Threads.RecorderRssThread
```

6.8 Scalable Resource Discovery System

The **SRDS** (Scalable Resource Discovery Service) provides several types of directory services to other modules of **XtreemOS**. Different interfaces and set of functionalities are defined for each client.

This first release provides to the **AEM** module the Resource Location Service and the Job Directory Service (**JDS**). The AEM interface returns a list of computing resources matching a specific resource query (in **JSDL** syntax), and the JDS interface allows to manage job information in a decentralized manner.

Each node within **XtreemOS** has to run an instance of the **SRDS**, that instance creating a few overlay networks (currently, two overlays are created).

1. Dynamically changing information is stored into one or more Distributed Hash Tables (**DHT**) by the SRDS; this version of SRDS exploits **ADS Bamboo**, a custom version of the Bamboo DHT. SRDS thus depends on the **ADS Bamboo** package. The DHT(s) is (are) run and initialized by the **SRDS** through a shell script, that controls the Bamboo configuration.

2. Static information about computing resources is retrieved through the **RSS** module (see Section 6.7). The RSS package is also an independent module, but in XtreemOS it is called directly by the **SRDS**, running within the same JVM. Thus **SRDS** and **RSS** depend on each one another.

6.8.1 Install SRDS

SRDS can be installed via `urpmi` with the following command:

```
urpmi srds
```

The package contains the SRDS only. In order to use it, it is also required to install the packages **ADS Bamboo** and **RSS**, which urmpi will propose to install if needed (see Sections 6.6 and 6.7).

6.8.2 Run SRDS

In order to run **SRDS** a few steps must be done.

Edit the file `XOSdConfig.conf`.

- In order to include the SRDS service bridge add the following line (also make sure that service number 13 isn't already occupied)
  `services.13=eu.xtreemos.xosd.srdsmng.service.SRDSMngHandler`

- Disable the use of SSL
  `useSSL=false`

- Define the network interface to use (if blank, eth0 will be used):
  `networkInterface=eth0`

- comment or delete the following line, in order to disable the ExecMngHandler service:
  `services.7=eu.xtreemos.xosd.execMng.service.ExecMngHandler`

Setup the ADS, RSS and Bamboo configuration files in /etc/xos/config as described in RSS and ADS Bamboo sections.

Overlay bootstrapping Remember to run the RSS recorder (tracker) in exactly one machine of your platform (you have chosen it in the RSS config file, see Section 6.7).

To do this, go to the XtreemOS jars directory and run

```
java -cp DIXIMain.jar:srds.jar:xtreemrss.jar:log4j-1.2.14.jar \
eu.xtreemos.ads.Threads.RecorderRssThread
```

Then run `xosd.sh`

6.9 XtreemFS

XtreemFS is the distributed file system of XtreemOS. It comprises three server modules:

Metadata and Replica Catalog (MRC)	MRCs are responsible for the file system metadata.
Object Storage Device (OSD)	OSDs store the content of files.
Directory Service (DIR)	The Directory Service acts as a global repository for information about OSDs, MRCs and file system volumes of an XtreemFS installation.

The file system is accessed by means of a client module:

Client/Access Layer (AL)	The Access Layer allows user processes to work with XtreemFS, via a POSIX interface. It is implemented as a user space module based on FUSE.

6.9.1 XtreemFS Installation

There are two different installation sources for XtreemFS: *RPM packages* and *source tarballs*.

Separate RPM packages exist for the server and the client components. To install the XtreemFS server components, execute

```
$> rpm -i XtreemFS-server.rpm
```

This will install an `init.d` script to set up the server components.

For the client components, execute

```
$> rpm -i XtreemFS-client.rpm
```

This will install the XtreemFS client to /usr/bin/xtreemfs

If you prefer using the source tarball, the first step is to build the components. The following third-party modules are required:

- Java Development Kit 1.6
- Apache Ant 1.6.5
- gmake 3.81
- gcc 4.1.2
- FUSE 2.6
- openssl-dev 0.9.8

To build the server components, make sure that JAVA_HOME and ANT_HOME are set. JAVA_HOME has to point to a JDK 1.6 installation, and ANT_HOME has to point to an Ant 1.6.5 installation.

Go to the top level directory and execute:

```
$> make
```

Once finished, the binaries can be used.

6.9.2 XtreemFS Security Preparations

In order to provide for security in an XtreemFS installation, services need to be equipped with X.509 certificates. Certificates are used to establish a mutual trust relationship between XtreemFS services, as well as between the XtreemFS client and XtreemFS services. Thus, in an XtreemOS environment, certificates have to be created for the services as a first step. This is done by creating a *Certificate Signing Request (CSR)* for the Root CA by means of the create-csr command. For further details, see Sec. 6.2.6.

Signed certificates and keys generated by the Root CA are stored locally in PEM format. Since XtreemFS services are currently not capable of processing PEM format, keys and certificates have to be converted to PKCS12 and Java Keystore format, respectively.

Each XtreemFS service needs a certificate and a private key in order to be run. Once they have created and signed, the conversion has to take place. Assuming that certificate/private key pairs reside in the current working directory for the Directory Service, an MRC and an OSD (`ds.pem`, `ds.key`, `mrc.pem`, `mrc.key`, `osd.pem` and `osd.key`), the conversion can be initiated with the following commands:

```
$> openssl pkcs12 -export -in ds.pem -inkey ds.key
   -out ds.p12 -name "DS"
$> openssl pkcs12 -export -in mrc.pem -inkey mrc.key
   -out mrc.p12 -name "MRC"
$> openssl pkcs12 -export -in osd.pem -inkey osd.key
   -out osd.p12 -name "OSD"
```

This will create three PKCS12 files (`ds.p12`, `mrc.p12` and `osd.p12`), each containing the private key and certificate for the respective service.

XtreemFS services need a *trust store* that contains all trusted Certification Authority certificates. Since all certificates created via the RCA have been signed by the XtreemOS CA, the XtreemOS CA certificate has to be included in the trust store. To create a new trust store containing the XtreemOS CA certificate, execute the following command:

```
$> keytool -import -alias xosrootca -keystore xosrootca.jks
   -trustcacerts -file
   /etc/xos/truststore/certs/xtreemos.crt
```

This will create a new Java Keystore `xosrootca.jks` with the XtreemOS CA certificate in the current working directory. The password chosen when asked will later have to be added as a property in the service configuration files.

Once all keys and certificates have been converted, the resulting files should be moved to `/etc/xos/xtreemfs/truststore/certs` as root:

```
# mv ds.p12 /etc/xos/xtreemfs/truststore/certs
# mv mrc.p12 /etc/xos/xtreemfs/truststore/certs
# mv osd.p12 /etc/xos/xtreemfs/truststore/certs
# mv xosrootca.jks /etc/xos/xtreemfs/truststore/certs
```

For details on XtreemFS security setup, please see The XtreemFS Installation and User Guide [6], Sections 2.2.2, 3.2.2, 3.2.7 and Appendix A.

6.9.3 XtreemFS Setup and Configuration

An XtreemFS installation requires at least one Directory Service, OSD and MRC to be running. In order to render XtreemFS accessible to user processes, a volume needs to be mounted via the Access Layer.

Default Setup. If installed from the RPM packages, default configuration files have already been created for the services. They are located in `/etc/xos/xtreemfs`. Three such configuration files exist for the different XtreemFS services: `dirconfig.properties` to configure a Directory Service, `mrcconfig.properties` to configure an MRC, and `osdconfig.properties` to configure an OSD. These are self-documenting Java properties files. If root access is granted, configuration parameters can be modified with an arbitrary text editor.

The following predefined ports are used for the services:

Directory Service	32638
MRC	32636
OSD	32640

Important: To guarantee that OSDs can be contacted by clients, it is necessary to ensure that OSDs register with their correct communication endpoints. XtreemFS will try to determine such endpoints automatically, but this often leads to wrong results. In order to ensure that OSDs are configured correctly, **it is necessary to manually edit the `uuid` property**.

```
# UUID for the OSD
# (property will become mandatory when UUIDs are resolved)
uuid = mydomain:32640
```

`mydomain` has to refer to either to an IP address, or to a fully-qualified distinguished name that can be resolved to an IP address. Defining the property ensures that `mydomain:32640` will be used as the endpoint at which the OSD is reachable.

The default configuration for the MRC and OSD assumes that the Directory Service runs on the local machine. The endpoint of the Directory Service to register at can be changed by modifying the following properties:

```
# Directory Service endpoint
dir_service.host = localhost
dir_service.port = 32638
```

For setting up a *secured* XtreemFS infrastructure, each service provides the following properties:

```
# specify whether SSL is required
ssl.enabled = true

# server credentials for SSL handshakes
ssl.service_creds = /etc/xos/xtreemfs/truststore/certs/\
service.p12
ssl.service_creds_pw = xtreemfs
ssl.service_creds_container = pkcs12

# trusted certificates for SSL handshakes
ssl.trusted_certs = /etc/xos/xtreemfs/truststore/certs/\
xosrootca.jks
ssl.trusted_certs_pw = xtreemfs
ssl.trusted_certs_container = jks
```

`service.p12` refers to the converted file containing the credentials of the respective service (see Sec. 6.9.2). Make sure that all paths and pass phrases (`xtreemfs` in this example) are correct.

In order to start and stop the services, three different shell scripts exist in `/etc/init.d`. All services can be started on the local machine by executing the following commands as root:

```
# /etc/init.d/xtreemfs-dir start
# /etc/init.d/xtreemfs-mrc start
# /etc/init.d/xtreemfs-osd start
```

Note that the Directory Service should be started first, in order to allow other services an immediate registration. Once a Directory Service and at least one OSD and MRC are running, XtreemFS is operational.

The services can be stopped by executing the following commands as root:

```
# /etc/init.d/xtreemfs-dir stop
# /etc/init.d/xtreemfs-mrc stop
# /etc/init.d/xtreemfs-osd stop
```

A detailed description of all configuration parameters can be found in Section 3.2.3 of The XtreemFS Installation and User Guide [6].

Automatic Setup. Alternatively, if XtreemFS has been delivered as a source tarball, an installation can also be set up automatically. The automatic setup allows you to customize your services for being run without root permissions. It is initiated by running the installation script located in the `install` directory:

```
$> cd install
$> ./install.sh
```

The script will guide you through the configuration of the different services. When the user is asked to enter the hostname or IP address for a service, **the name or address proposed by the installer might not be correct!** The user has to ensure that all machines in the XtreemFS installation can resolve the name.

Once all configuration parameters have been confirmed, go the target directory of the installation and execute:

```
$> ./start.sh
```

This will automatically set up the services as configured before. In order to shut down all XtreemFS services, execute

```
$> ./stop.sh
```

in the same directory.

6.10 XOSAGA

XOSAGA is available in XtreemOS as several rpm packages:

libsaga-devel contains everything to develop SAGA applications.

libsaga contains all libraries to run SAGA applications.

saga contains some example SAGA programs and environment settings.

xosaga contains XtreemOS-specific additions to SAGA.

Installing XOSAGA can be done using urpmi:

```
$> urpmi xosaga
```

You will be given a choice between the 'libsaga' and the 'libsaga-devel' package. Choose the 'libsaga' package if you only want to *run* SAGA applications (e.g. on an XtreemOS node). Choose the 'libsaga-devel' package if you also want to develop XOSAGA applications on your machine.

6.11 LinuxSSI

A LinuxSSI cluster can act as a resource in XtreemOS. Please, do not use it as a server node. Beware that all your cluster nodes must be on the same physical network. There must be no router between them.

The following steps have to be done on each cluster's node:

1. Install XtreemOS as described in section 3. Do not forget to select LinuxSSI.

2. Configure the boot-loader to have different *node id* for each node of your cluster. Node id are numerical numbers between 1 and 254. To set the *node id*, you need to add on the kernel command line `node_id=`*X*, with *X* the *node id*. This configuration can be done during the installation process or after by modifying file `/boot/grub/menu.lst`.

   You must also check that the LinuxSSI kernel is the default kernel to boot.

3. Once all nodes have been installed, check the DNS configuration or edit `/etc/hosts` to be able to contact each cluster's node from any cluster's node.

4. You need to create file `/etc/xos/linuxssi/nodes` and to fill it with the hostname of each cluster's node with one node per line. The first node will act as the *head* node.

   ```
   root# emacs /etc/xos/linuxssi/nodes
   clusternode1
   clusternode2
   clusternode3
   ...

   Quit emacs.
   ```

5. Configure ssh so that you can connect from head node to other nodes as root.

The following steps have to be done **only on the *head* node**:

1. Install package kanif

```
root# urpmi kanif
```

2. Install package nfs-utils

   ```
   root# urpmi nfs-utils
   ```

It is now time to have all your cluster node booted on LinuxSSI kernel. Connect as root to the head node and start LinuxSSI:

```
root# start_linuxssi
```

This command will

- configure some NFS exports (`/etc/xos`, `/home`, `/root`, `/tmp`, `/usr/local`, `/var`),
- start LinuxSSI (`krgadm cluster start`),
- start the AEM-LinuxSSI listener,
- switch the head node to runlevel 4,
- make other LinuxSSI cluster's nodes run `start_linuxssi_as_worker`. `start_linuxssi_as_worker` will
  - switch the node to runlevel 3,
  - mount the NFS sharing,
  - start the AEM-LinuxSSI-dispatcher.

You can check that LinuxSSI is running by invoking `cat /proc/cpuinfo` that must show information about the CPU of all your cluster nodes.

Then, you can run the XtreemOS resource configuration on the *head* node as if it is a single node, as explained in section 5.

Once you have finished, and *xosd* is running, run:

```
root# start_linuxssi_scheduler.sh
root# xosd_linuxssi_capabilities
```

The first command will start the legacy LinuxSSI scheduler. You can configure an personalized one if you prefer (see Section 7.8). The second command will ensure that Xosd has and will give good LinuxSSI capabilities for jobs to be distributed and migrated on the cluster.

7 Using XtreemOS

7.1 Virtual Organization Management

7.1.1 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 7.1.4.

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

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

```
get-xos-cert <cdaserverFQHN>:<port>
```

Figure 10: Obtaining an XOS-certificate from a CDA server

Where <`cdaFQHN`> is to be replaced with the Fully-Qualified Host Name or IP address for the CDA server in your particular XtreemOS Grid. The <`port`> should be replaced with the numeric port that the CDA server is listening on. Your Grid administrator can provide these details.

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 11: Viewing an XOS-Certificate

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

2) The user credentials can be generated via the VOLife webapp in two steps, as described in section 7.1.4, by following the operations 'Generate new keypair' and 'Generate an XOS-Cert'.

7.1.2 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 application 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 12 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 12: Examining an application 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 application certificate from the request. Figure 13 shows the step needed to create an application 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 13: Signing a host CSR file, creating an application certificate.

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

7.1.3 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=test.testcom.com
mrc.port=32636
```

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

7.1.4 Using VOLife

Using the web frontend of VOLife

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://volifeservice:8080/volifecycle/
```

- By default, Tomcat is listening on port 8080 for accepting http requests. If you access the above link with failure message, check the listening port of Tomcat with the following command:

```
cat /usr/share/tomcat5/conf/server.xml|grep "Connector port" -m 1
```

The result should show the default service port of Tomcat, such as

```
<Connector port="8080" maxHttpHeaderSize="8192"
```

and replace the above url with the correct port when neccessary.

- 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.

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. Note that it is only used in experimental environments 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 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:

```
Usage: java XVOMSUtil [options]
[options]
        -create-user <user_name> <password> <first_name> <last_name> <organiz
                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_ow
                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_day
                Generate XOS-Cert for a given user in a VO
```

7.1.5 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 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:

- **add_resource_to_vo** *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.

- **dixi_test -RCA advo** — set the current node to be a member of the test VO. The result of the script invocation will be a new VO-related attribute certificate placed into the folder defined in **RCAClientConfig.conf** as **resVOAttributeCertIncoming**. To install it, simply move it to the same folder that contains the machine identity certificate.

- **dixi_test -RCA rvo** *VO_id* — remove the current node from a membership in the VO with *VO_id* for its id.

- **dixi_test -RCA rdvo** — remove the current node from a membership in the test VO.

- **request_vo_certificate** *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.

- **dixi_test -RCA cdvo** — request a VO-related attribute certificate of the test VO 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:

- **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 14 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.

7.1.6 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.

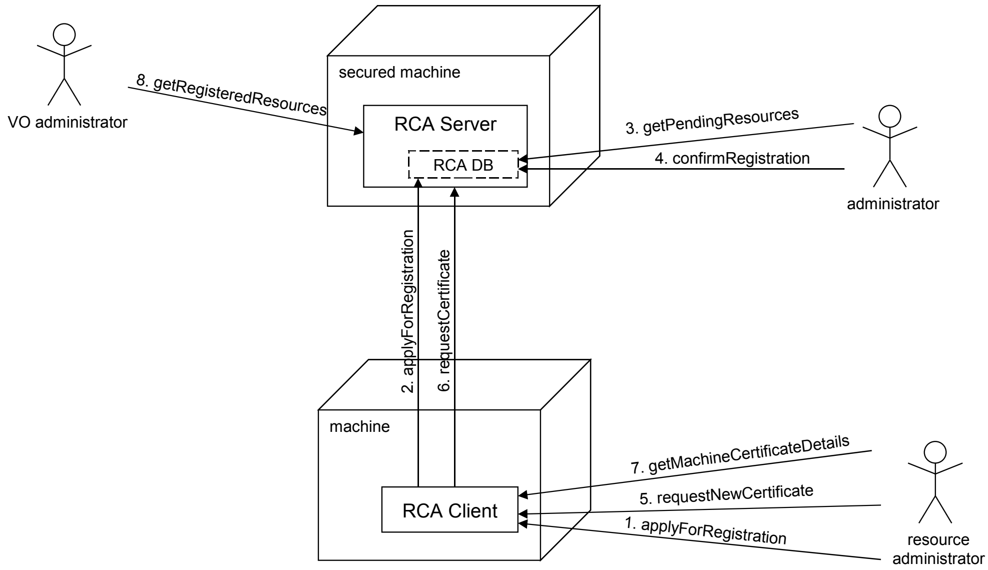

Figure 14: Interactions between resource admin and RCA server

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** ⟨*polcyId*⟩ 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** ⟨*xacmlRule*⟩ **-pid** ⟨*policyId*⟩ 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 16.

- **xremoveRuleFromPolicy -rid** ⟨*ruleId*⟩ **-pid** ⟨*policyId*⟩ removes rule identified by **ruleId** from policy with **policyId**.

- **xremovePolicy -pid** ⟨*policyId*⟩ Removes policy identified by **policyId** from policy storage

- **xaddPolicy -policy** ⟨*xacmlPolicy*⟩ adds policy residing localy on the path **xacmlPolicy** into VOPS policy storage. See also example on picture 15.

- **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 15. Other policies can be added using PAP and commands listed above. More refined rule, which can be injected into policy, is listed in figure 16.

7.2 Application Execution Management

The interaction with the AEM occurs through the use of **xconsole_dixi** which, in turn, uses DIXI's XATI to communicate with the services.

```
<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 15: A default XACML policy. This policy permits all actions to all users on a resurce in query.

7.2.1 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 17 shows an example of a JSDL document, which describes a job with a single executable, defines its standard output and standard error stream files, and has no specific resource requirements.

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 18 shows a sample JSDL containing the resource requirements.

```
  <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:globalPrimaryVOName"
     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 16: 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).

7.2.2 Using AEM through XConsole

The **xconsole_dixi** script is a part of 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/XOSdConfig.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.

```
<?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 17: A typical JSDL for submitting a job with no resource requirements.

- `xps j jobID` — Shows the info for the job with id jobID.
- `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 fits into the jsdl requirements.

7.3 ADS_Bamboo

The ADS_Bamboomodule is not used alone, it is launched by the SRDS module and exploited by it.

7.4 RSS

Using RSS alone is possible, but not currently described and configured.

```
<?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 18: A sample JSDL for limiting the resource requirements for the job.

7.5 SRDS

Using the SRDS module is currently done through the AEM module.

7.6 XtreemFS

To render XtreemFS usable, a volume has to be created, if necessary, and mounted. An in-depth user guide and description of all tools can be found in The XtreemFS Installation and User Guide [6].

7.6.1 Security Preparations

As a first step, it is necessary to generate a private key and a certificate for the local XtreemFS client. This will allow the client to connect to a secured XtreemFS installation. A Certificate Signing Request has to be issued to the RCA in a similar way as it is done for the services. More details on this are given in Sec. 7.1.5. The resulting credentials (`client.pem`, `client.key`) have to be bundled in a PKCS12 file, which can be done as follows:

```
$> openssl pkcs12 -export -in client.pem -inkey client.key
   -out client.p12 -name "CLIENT"
```

The resulting file `client.p12` should be moved to `/etc/xos/xtreemfs/truststore/certs` (root permissions required):

```
# mv client.p12 /etc/xos/xtreemfs/truststore/certs
```

The credentials file will later be used to mount an XtreemFS volume.

7.6.2 Creating a New Volume

Before XtreemFS can be mounted, the first step is to create a new volume at the MRC. The `xtfs_mkvol` command supports the creation of volumes with different default settings. With an MRC running on the local host, a volume could be created as follows:

```
$> xtfs_mkvol -a 2 -c /etc/xos/xtreemfs/truststore/certs/client.p12
   https://localhost:32636/MyVolume
```

This command creates a new volume at an MRC that runs on the local host at port 32636 and enforces a POSIX access permission model on files. In order to connect to a secured MRC, it uses the given certificate. For further information on command line arguments, use the `--help` option.

Note that there are further commands that relate to the management of volumes. `xtfs_lsvol` prints a list of all volumes stored on an MRC, and `xtfs_rmvol` deletes an existing volume, including all data stored in it.

7.6.3 Loading the FUSE Driver

Before mounting XtreemFS volumes, it might be necessary to add the FUSE driver to the kernel. In order to do this, execute the following statement as root:

```
# modprobe fuse
```

7.6.4 Mounting an XtreemFS Volume

Once the FUSE driver has been loaded, the previously created volume can be mounted by executing `xtfs_mount`, e.g. as follows:

```
$> xtfs_mount -o dirservice=http://localhost,
   volume_url=https://localhost:32636/MyVolume,ssl_cert=
   /etc/xos/xtreemfs/truststore/certs/client.p12 ~/mountpoint
```

`mountpoint` refers to the mount point in the local directory tree to which XtreemFS is mounted.

`volume_url` points to the URL of the volume, i.e.
`<protocol>://<MRC endpoint>/<volume name>`.

`ssl_certs` defines the location of the client credentials, which is needed to establish X.509-based SSL connections to XtreemFS services.

For further details on command line arguments, execute `xtreemfs --help`.

7.6.5 Unmounting an XtreemFS Volume

A mounted XtreemFS volume can be unmounted using the `xtfs_umount` script.

```
$> xtfs_umount ~/mountpoint
```

`mountpoint` refers to the mount point in the local directory tree to which XtreemFS was mounted using `xtfs_mount`.

7.6.6 Monitoring XtreemFS Services

Each XtreemFS service provides a monitoring interface. To retrieve information about a service, just start an arbitrary web browser and browse to `<protocol>://<serviceHost>:<servicePort>`, where `protocol` refers to `http` or `https`, depending on whether security is enabled, `<serviceHost>` refers to the hostname or IP address of the service, and `<servicePort>` refers to the port number the service is listening at. Note that if security is enabled, the browser needs a trusted certificate, e.g. the client certificate.

7.6.7 Known Bugs and Issues

Some issues have been identified that haven't yet been solved:

- FUSE does not support mmap in connection with direct I/O. In order to get applications running on XtreemFS that rely on mmap, volumes have to be mounted without using the FUSE option `-o direct_io`. However, this might lead to inconsistencies if different clients concurrently work on the same file, as requests might be serviced from the local page cache.

- When run on older Linux kernels (2.6.18 or earlier), the access layer might fail to enforce access control when changing to a directory. The problem can be solved by using the `-o default_permissions` parameter when mounting the volume.

7.7 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.

7.7.1 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. Note that to mount XtreemFS volumes, your user should be part of the group ’fuse’. This can be done using the gpasswd command:

```
$> gpasswd -a <username> fuse
```

Now 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
```

7.8 LinuxSSI

7.8.1 Using kDFS

The *kernel Distributed File System* aims at federating storage resources within a cluster in order to provide an integrated cluster file system for High Performance Computing. Design, implementation and current status of the first prototype is addressed in a distinct deliverable [3]. This section relates procedures to exploit a kDFS file system. The first paragraph addresses the `mkfs.kdfs` command, the second focuses on mounting a kDFS partition. Note: Kerrighed has to be started to set/exploit a kDFS structure.

Formatting a kDFS partition

Associated to kDFS, a dedicated command, mkfs.kdfs has been implemented. This command formats a directory which can be used afterward in the kDFS physical structure. This command is released with the current kDFS prototype. Its syntax is the following one:

```
mkfs.kdfs DIRECTORY_PATHNAME ROOT_NODEID
```

`DIRECTORY_PATHNAME`: Absolute path to store kDFS meta-data an data.
`ROOT_NODEID`: kDFS root node id, localization of the kDFS root meta-data file.

`mkfs.kdfs` creates the kDFS "superblock" file for the node. This file is stored on the local native file system at `DIRECTORY_PATHNAME` if `KERRIGHED_NODEID` equals `ROOT_NODEID`, `mkfs.kdfs` creates the root meta-file which is the clusterwide entry point for the kDFS physical structure.

The table 2 describes creation of a kDFS structure distributed between two nodes:

On node A: (kerrighed nodeid = 1)	on Node B: (kerrighed nodeid = 2)
`mkfs.kdfs /KDFS 1`	`mkfs.kdfs /ANOTHER-KDFS 1`
Create kDFS local superblock Create kDFS root meta-file	Create kDFS local superblock

Table 2: Create a kDFS structure between two nodes

Mounting a kDFS partition

The `mount` command is the traditional one. We have still not extended it with specific kDFS parameters. Thus, for the moment, There are few limitations:

- One kDFS partition per node
- Only one mount point per node
- Diskless mechanisms not available

Users can mount a kDFS partition by the following command:

```
mount -t kdfs ALLOCATED_DIRECTORY MOUNT_POINT
```

`ALLOCATED_DIRECTORY`: native file system directory formatted with mkfs.kdfs `MOUNT_POINY`: traditional mount point.

The table 3 describes kDFS mounting procedure from two nodes:

On node A: (kerrighed nodeid = 1)	on Node B: (kerrighed nodeid = 2)
`mount -t kdfs /KDFS /mnt/kfds`	`mount -t kdfs /ANOTHER-KDFS /mnt/kdfs`
`/mnt/kdfs` is now a global kDFS namespace for both nodes	

Table 3: Mount kDFS partitions

Finally, kDFS documentation is available on the Kerriged web site. Please visit http://www.kerrighed.org/wiki/index.php/KernelDevelKdFS

7.8.2 Customizable scheduler

In this document, we are only describing how to enable “Mosix Load Balancer”, a CPU-load-balancing scheduling policy which uses LinuxSSI Pluggable Probes and Scheduling Policies (PlugProPol) infrastructure to migrate processes among LinuxSSI cluster nodes. Additional data on LinuxSSI and PlugProPol framework (i.e. the description PlugProPol commands and API, the architecture, etc.) can be found in [5] and [1].

Before you start loading “Mosix Load Balancer” scheduling policy and its probes, you have to make sure that you have the “config” directory created in the root directory of LinuxSSI system image (i.e. the operating system directory structure that is deployed to the diskless LinuxSSI clients. Usually). This a directory, to which the configfs file-system for controlling PlugProPol infrastructure is mounted. Usually, the LinuxSSI system image is located in */NFSROOT/kerrighed* directory, therefore you have to check if you have the */NFSROOT/kerrighed/config* directory created.

Since the PlugProPol framework takes care of executing the operations cluster-wide, the following steps have to be executed on a single node of LinuxSSI cluster:

1. first, load all the necessary probes (i.e. Mosix probe and Migration probe). These probes provide Mosix Load Balancer the necessary data for deciding when to perform migration:

   ```
   mkdir "$MOSIX_PROBE_PATH"
   mkdir "$MIGRATION_PROBE_PATH"
   ```

2. create Mosix Load Balancer:

   ```
   mkdir -p "$POLICY_PATH"
   ```

3. load all the necessary filters (i.e. Threshold filter and Remote cache filter). In PlugProPol framework, filters are used either for filtering the data that come from the probes or for storing it. Besides loading the filters you also have to set their parameters:

   ```
   mkdir "${POLICY_PATH}/local_load/freq_limit_filter"
   echo $MIGRATION_INTERVAL > \
     "${POLICY_PATH}/local_load/freq_limit_filter/min_interval"
   mkdir "${POLICY_PATH}/local_load/freq_limit_filter/ \
     threshold_filter"
   echo $LOAD_THRESHOLD > "${POLICY_PATH}/local_load/ \
     freq_limit_filter/threshold_filter/threshold"
   mkdir "${POLICY_PATH}/remote_load/remote_cache_filter"
   echo $POLL_INTERVAL > "${POLICY_PATH}/remote_load/ \
     remote_cache_filter/polling_period"
   ```

4. connect Mosix Load Balancer to the probes so it can start collecting data from them:

   ```
   mkdir "${POLICY_PATH}/single_process_load/remote_cache_filter"
   ln -s "${MIGRATION_PROBE_PATH}/last_migration" \
     "${POLICY_PATH}/local_load/freq_limit_filter/last_event/migration"
   ln -s "${MOSIX_PROBE_PATH}/process_load" \
     "${POLICY_PATH}/process_load/mosix"
   ln -s "${MOSIX_PROBE_PATH}/norm_upper_load" \
     "${POLICY_PATH}/remote_load/remote_cache_filter/mosix_upper"
   ln -s "${MOSIX_PROBE_PATH}/norm_single_process_load" \
     "${POLICY_PATH}/single_process_load/remote_cache_filter/mosix"
   ln -s "${MOSIX_PROBE_PATH}/norm_mean_load" \
   ```

```
"${POLICY_PATH}/local_load/freq_limit_filter/threshold_filter/ \
mosix_mean"
```

5. make Mosix Load Balancer in charge of migrating all the processes on a in LinuxSSI cluster

   ```
   echo 1 > "${SCHEDULER_PATH}/process_set/handle_all"
   ```

After the Mosix Load Balancer has been successfully loaded and configured it takes care of CPU-load balancing by itself without user intervention. It automatically migrates processes from the more utilised nodes of the LinuxSSI cluster to the less utilised ones. By doing this, it tries to maintain the CPU utilisation around the nodes as even as possible.

For a clearer understanding you can find a complete Bash script for enabling Mosix Load Balancer in LinuxSSI. You have to execute this script on a single node of a LinuxSSI cluster, immediately after the Kerrighed cluster has was started:

```
# Round robin balancing with remote clone
SCHEDULER_PATH=/config/krg_scheduler/schedulers/rr

mkdir -p "$SCHEDULER_PATH/round_robin_balancer" && \
echo 1 > "$SCHEDULER_PATH/process_set/handle_all"

# Dynamic Mosix-like CPU load balancing with migration
SCHEDULER_NAME=mosix
# 2 seconds between each migration
MIGRATION_INTERVAL=2000000000
# Refresh remote node loads every second
POLL_INTERVAL=1000

SCHEDULER_PATH="schedulers/$SCHEDULER_NAME"
POLICY_PATH="${SCHEDULER_PATH}/mosix_load_balancer"
MOSIX_PROBE_PATH=probes/mosix_probe
MIGRATION_PROBE_PATH=probes/migration_probe

cd /config/krg_scheduler && \
mkdir "$MOSIX_PROBE_PATH" && \
mkdir "$MIGRATION_PROBE_PATH" && \
```

```
mkdir -p "$POLICY_PATH" && \
mkdir "${POLICY_PATH}/local_load/freq_limit_filter" && \
echo $MIGRATION_INTERVAL > "${POLICY_PATH}/local_load/ \
  freq_limit_filter/min_interval" && \
mkdir "${POLICY_PATH}/local_load/freq_limit_filter/threshold_filter" && \
v=‘< "${MOSIX_PROBE_PATH}/norm_single_process_load/value"‘ && \
echo ‘echo "$v + $v / 2" | bc‘ > "${POLICY_PATH}/local_load/ \
  freq_limit_filter/threshold_filter/threshold" && \
mkdir "${POLICY_PATH}/remote_load/remote_cache_filter" && \
echo $POLL_INTERVAL > "${POLICY_PATH}/remote_load/remote_cache_filter/ \
  polling_period" && \
mkdir "${POLICY_PATH}/single_process_load/remote_cache_filter" && \
ln -s "${MIGRATION_PROBE_PATH}/last_migration" "${POLICY_PATH}/local_load/ \
  freq_limit_filter/last_event/migration" && \
ln -s "${MOSIX_PROBE_PATH}/process_load" "${POLICY_PATH}/process_load/ \
  mosix" && \
ln -s "${MOSIX_PROBE_PATH}/norm_upper_load" "${POLICY_PATH}/remote_load/ \
  remote_cache_filter/mosix_upper" && \
ln -s "${MOSIX_PROBE_PATH}/norm_single_process_load" "${POLICY_PATH}/ \
  single_process_load/remote_cache_filter/mosix" && \
ln -s "${MOSIX_PROBE_PATH}/norm_mean_load" "${POLICY_PATH}/local_load/ \
  freq_limit_filter/threshold_filter/mosix_mean" && \
echo 1 > "${SCHEDULER_PATH}/process_set/handle_all"
```

7.8.3 Adding and removing node(s) in the cluster

Design, implementation and current status of the first prototype is addressed in a distinct deliverable [2].

The command used to manage node(s) addition and node(s) removal in the cluster is krgadm.

krgadm provides to the user a unique command to monitor the status of the cluster, to start a cluster, to make node(s) join or leave the cluster.

Nodes are identified either by their nodes' id, either by their MAC addresses.

Administrator can start a cluster with all the available nodes (Kerrighed module loaded) with the following command:

```
# krgadm cluster start
```

To add some nodes in a running cluster, for instance nodes 16 and 18, the following command must be used by the administrator:

```
# krgadm nodes add -n16:18
```

To remove some nodes in a running cluster, for instance nodes 21 and 34, the following command must be used by the administrator:

```
# krgadm nodes del -n21:34
```

More information can be found with `man krgadm` or `krgadm -h`. Some options described in `krgadm` manual or help are not yet implemented.

7.8.4 Checkpointing/Restarting application

After taking the initial checkpoint, a directory will be created under /var/chkpt. The ID of the checkpointed application serves as directory name.

Second, a so called capability has to be set. Certain system features can be en/disabled with capabilities. Such a capability has to be switched on for checkpointing to work. The command for enabling the checkpointing functionality is:

```
# krgcapset -d +CHECKPOINTABLE
```

This command must be executed in the shell on which the process to be checkpointed will be started.

A checkpoint is created by issuing the following command:

```
# checkpoint PID
```

After successfully taking a checkpoint, a directory (checkpointed process' PID as directory name) should have been created under /var/chkpt. The following files will be created in this directory each including the serial number (SN) of the checkpoint in their name, e.g. '4711' for the file names mentioned below:

- *description_v4711.txt* (ascii file): short overview description of all files belonging to the checkpoint.

- *global_v4711.bin* (binary file): app_kddm_object KDDM object values as the applicationID, the serial number of the checkpoint and the kerrighed node mask.
- *node_5_v4711.bin* (binary file): description of local tasks involved in the checkpoint. The sample file belongs to node 5.
- *task_234_v4711.bin* (binary file): per task (e.g. PID='234') kernel structures such as registers, stack, signal mask, ...
- *task_mm_234_v4711.bin* (binary file): all pages of the process address space.

Repeated checkpointing of an application results in multiple files with the same base name but an SN increased by one at each time acheckpoint is taken. Checkpoints taken at different times can thus be distinguished.

An application is restarted by executing the following command:

```
# restart PID SN
```

Providing a SN allows to specify one out of many checkpoints taken during application execution.

For an application consisting of two or more processes, each of them having parent-child relationship, the PID of the common ancestor has to be provided to the restart command.

7.8.5 Integration of LinuxSSI checkpointing with XtreemOS grid checkpointing

In the current release of LinuxSSI (0.9 beta) LinuxSSI has been prepared for grid checkpointing to work. In a later version it will be possible to call these LinuxSSI checkpoint and restart primitives from the AEM XConsole.

The intended command syntax for checkpointing is the following:

```
# xcheckpoint jobID
```

The intended command syntax for restart is the following:

```
# xrestart jobID version
```

7.8.6 Integration of LinuxSSI process management extensions into XtreemOS

AEM must be aware of all processes running on a LinuxSSI cluster. In order to maintain the SSI semantic AEM will be executed on a dedicated master node in the cluster. This can lead to ignoring certain process events on slave nodes and misinterpretation of certain process events on the master node. To overcome this shortcoming a process event propagation mechanism has been realised. The administrator must ensure that each slave node executes a client program that propagates relevant local process events (initial forks and final exits) to the master node. In a future version of AEM the ExecMng will receive those propagated events and take them into account. Thus, AEM has knowledge only of those process events that comply with the SSI semantic.

8 Annex I: Enabling kernel connectors

Kernel connectors need to be compiled into the kernel, as using the cn module is not enough. In this following steps we describe the steps that can be followed in order to have a connector enabled kernel

Debian distribution To compile and install the kernel with connectors, follow the next steps:

1. Download the source code for the kernel that we want to install. In our case we used the lastest version available in the repository configured in our server:
    - bash:$ sudo apt-get install linux-source
2. Go to the directory file where you have stored the kernel sources, untar them and make the menuconfig. To do this you can follow the next steps:
    - Check that you have the libncurses and qt3 installed. If not:
        - bash:/usr/src$ sudo apt-get install libncurses5-dev
        - bash:/usr/src$ sudo apt-get install libqt3-dev
    - bash:$ cd /usr/src
    - bash:/usr/src$ sudo tar jxvf linux-source-2.6.8.1.tar.bz2
    - bash:/usr/src$ cd linux-source-2.6.8.1
    - bash:/usr/src/linux-source-2.6.8.1$ sudo make menuconfig
    - In the menuconfig go to the "Device Drivers →" section and mark with "*" the option "Connector - unified userspace ←→ kernelspace linker"
    - make
3. Create the debian package to install the kernel:
    - Check that you have the build-essential and kernel-package installed. If not:
        - bash:/usr/src/linux-source-2.6.8.1$ sudo apt-get install build-essential
        - bash:/usr/src/linux-source-2.6.8.1$ sudo apt-get install kernel-package

- bash:/usr/src/linux-source-2.6.8.1$ sudo make-kpkg clean
- bash:/usr/src/linux-source-2.6.8.1$ sudo make-kpkg –append-to-version=.XXXX –initrd kernel_image
- Where the XXXX is the name that will be appended at the end of the kernel name
- In the /usr/src you will have a debian package like: kernel-image-2.6.8.1.XXXX_10.00.Custom_i386.deb

4. Install the kernel:

   - bash:/usr/src$ sudo dpkg -i kernel-image-2.6.8.1.XXXX_10.00.Custom_i386.deb

Mandriva distribution In the case that you are using the mandriva distribution you have to follow the nexts steps:

1. Download the source code for the kernel that we want to install in the "/usr/src" directory:

   - bash:$ cd /usr/src
   - bash:/usr/src$ wget http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.18.3.tar.bz2

2. Go to the directory file where you have stored the kernel sources, untar them and make the menuconfig. To do this you can follow the next steps:

   - bash:$
   - bash:/usr/src$ tar xjf linux-2.6.18.3.tar.bz2
   - bash:/usr/src$ ln -s linux-2.6.18.3 linux
   - bash:/usr/src$ cd linux
   - bash:/usr/src/linux$ make menuconfig
   - In the menuconfig go to the "Device Drivers →" section and mark with "*" the option "Connector - unified userspace ←→ kernelspace linker"
   - make

3. Create the rpm that will be used to install the kernel:

   - bash:/usr/src/linux$ make rpm

- After the successful kernel build, a src.rpm and an rpm package have been created. The *src.rpm package can be found in the /usr/src/rpm/SRPMS/ directory. The rpm package can be found in /usr/src/rpm/RPMS/i386/, /usr/src/rpm/RPMS/i586/, /usr/src/rpm/RPMS/ /usr/src/rpm/RPMS/x86_64/, etc., depending on your architecture. On my system it was located in /usr/src/rpm/RPMS/i386/.

4. Installing the kernel (supose that the *src.rpm file is kernel-2.6.18.3default-1.i386.rpm):

   - bash:/usr/src/linux$ cd /usr/src/rpm/RPMS/i386/
   - bash:/usr/src/linux$ rpm -ivh kernel-2.6.18.3default-1.i386.rpm
   - bash:/usr/src/linux$ mkinitrd /boot/initrd-2.6.18.3-default.img 2.6.18.3-default

References

[1] XtreemOS consortium. Design and implementation of a customizable scheduler. Deliverable D2.2.6, November 2007.

[2] XtreemOS consortium. Design and implementation of basic reconfiguration mechanisms. Deliverable D2.2.4, November 2007.

[3] XtreemOS consortium. Design and implementation of high performance disk input/output operations in a federation. Deliverable D2.2.5, November 2007.

[4] XtreemOS consortium. Prototype of the basic version of linux-xos. Deliverable D2.1.4, November 2007.

[5] XtreemOS consortium. Prototype of the basic version of LinuxSSI. Deliverable D2.2.7, October 2007.

[6] Björn Kolbeck, Jan Stender, and Felix Hupfeld. The XtreemFS Installation and User Guide. http://www.xtreemfs.org/xtfs-guide.pdf.