Project no. IST-033576

XtreemOS

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

XtreemFS and Object Sharing Service: Second Prototype D3.4.4

Due date of deliverable: 30-NOV-2008
Actual submission date: 14-NOV-2008

Start date of project: June 1st 2006

Type: Deliverable
WP number: WP3.4

Responsible institution: ZIB
Editor & and editor's address: Felix Hupfeld
Zuse Institute Berlin
Takustr. 7
14195 Berlin
Germany

Version 1.0 / Last edited by Björn Kolbeck / 13-NOV-2008

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

Revision history:

Version	Date	Authors	Institution	Section affected, comments
0.1	13.10.08	Björn Kolbeck, Jan Stender, Felix Hupfeld	ZIB	Initial version of XtreemFS User Guide
0.2	23.10.08	Kim-Thomas Möller	UDUS	Initial version of OSS User Guide
0.3	24.10.08	Jan Stender	ZIB	Minor fixes
0.4	04.11.08	Björn Kolbeck, Jan Stender	ZIB	Incorporated Reviewer's comments, Minor fixes
0.5	12.11.08	Marc-Florian Müller	UDUS	Incorporated Reviewer's comments
0.6	13.11.08	Björn Kolbeck, Jan Stender	ZIB	Final draft

Reviewers:
Mathijs den Burger (VUA), Adolf Hohl (SAP)

Tasks related to this deliverable:

Task No.	Task description	Partners involved$^{\circ}$
T3.4.1	XtreemFS File Access	CNR*, BSC, ZIB
T3.4.2	XtreemFS Metadata Server	ZIB*
T3.4.4	XtreemFS Pattern-Aware Data Access	BSC*, CNR
T3.4.5	Object Sharing Service	UDUS*
T3.4.6	XtreemFS client	NEC*, UDUS

$^{\circ}$This task list may not be equivalent to the list of partners contributing as authors to the deliverable
*Task leader

Contents

Chapter 1

Executive Summary

This documents contains the two user guides for The XtreemFS File System and for the Object Sharing Service (OSS).

The XtreemFS user guide describes the release 0.10 of XtreemFS which is publicly available under the GPL license. The current release presents a fully functional POSIX compatible distributed file system. Advanced features like encryption and authentication between components using SSL and striping over multiple storage servers are implemented and well-tested.

The OSS user guide describes the API as well as the installation and usage of release 0.2 of the Object Sharing Service which is publicly available under the GPL license.

Chapter 2

The XtreemFS User Guide

2.1 Quick Start

This is the very short version to help you set up a local installation of XtreemFS.

1. Download XtreemFS RPMs/DEBs and install

   (a) Download the RPMs or DEBs for your system from the XtreemFS website

   (b) open a root console (`su` or `sudo`)

   (c) install with `rpm -Uhv xtreemfs-client-0.10.0.rpm xtreemfs-server-0.10.0.rpm`

2. Start the Directory Service:
   `/etc/init.d/xtreemfs-dir start`

3. Start the Metadata Server:
   `/etc/init.d/xtreemfs-mrc start`

4. Start the OSD:
   `/etc/init.d/xtreemfs-osd start`

5. If not already loaded, load the FUSE kernel module:
   `modprobe fuse`

6. Depending on your distribution, you may have to add users to a special group to allow them to mount FUSE file systems. E.g.: In openSUSE

users must be in the group `trusted`, in Ubuntu in the group `fuse`. You may need to logout and login again for the new group membership to become effective.

7. You can now close the root console and work as a regular user.

8. Wait a few seconds for the services to register at the directory service. You can check the registry by opening the DIR status page in your favorite web browser http://localhost:32638.

9. Create a new volume with a stripe size of 256kB:
   `xtfs_mkvol -p RAID0,256,1 http://localhost/myVolume`

10. Create a mount point:
    `mkdir ~/xtreemfs`

11. Mount XtreemFS on your computer:

    ```
    xtfs_mount -o dirservice=http://localhost, \
    volume_url=http://localhost/myVolume ~/xtreemfs
    ```

12. Have fun ;-)

13. To un-mount XtreemFS:
    `xtfs_umount ~/xtreemfs`

You can also mount this volume from other computers. First make sure that the ports 32636, 32638 and 32640 are open for incoming TCP connections. You must also specify a hostname that can be resolved by the remote machine! Finally, you have to use `http://`*`hostname`* instead of `http://localhost` when mounting.

2.2 What is XtreemFS

2.2.1 About XtreemFS

With XtreemFS you are about to install a modern *distributed file system*. As a distributed file system, XtreemFS stores your file data on several servers and you can simply scale your file system by adding more hosts. XtreemFS is a full-featured file system that supports the full POSIX file interface, including *extended attributes* (xattrs). In case of concurrent access by several distributed programs, XtreemFS provides you currently with NFS close-to-open consistency.

XtreemFS has been designed for deployment in *wide-area environments* connected by the Internet. This means that it allows you to mount an XtreemFS volume from any location, given the right permissions; but it also implies that file system installations can span multiple locations or data centers.

In a normal UNIX environment, XtreemFS has full permission and *POSIX ACL* support. XtreemFS can also be integrated into *X.509-based security architectures*. Access policies (as well several other policies) are pluggable and can be easily extended. If you deploy XtreemFS as part of an *XtreemOS* installation, you will benefit from its transparent integration with the XtreemOS *Virtual Organization (VO)* infrastructure in the form of dynamic user mappings and automatic mounting of home volumes.

If you need *high-performance* access to your files, XtreemFS can help you with support for *file striping*: XtreemFS can store a file across several storage servers and *access* the parts *in parallel*. The size of an individual stripe and the number of storage servers used can be configured on a per-file or per-directory basis.

2.2.2 XtreemFS Architecture

XtreemFS implements an *object-based file system architecture* (Fig. 2.1). The name of this architecture comes from the fact that an object-based file system splits file content into a series of fixed-size *objects* and stores them on its storage servers. In contrast to block-based file systems, the size of such an object can vary from file to file.

The *metadata* of a file (such as the file name or file size) is stored separate from the file content on a Metadata server. This metadata server organizes file system metadata as a set of *volumes*, each of which implements a separate file system namespace in form of a directory tree.

The Components of XtreemFS

An XtreemFS installation contains three types of servers that can run on one or several machines (Fig. 2.1):

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

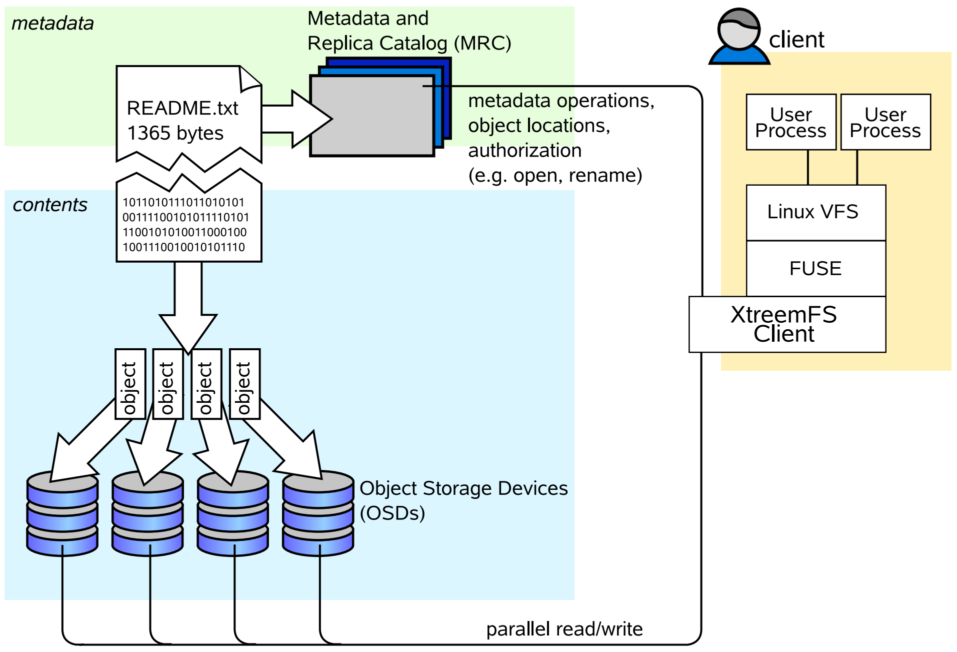


Figure 2.1: The XtreemFS architecture and components.

- MRC - Metadata and Replica Catalog
  The 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.
- OSD - Object Storage Device
  An OSD stores arbitrary objects of files; clients read and write file data on OSDs.

These servers are connected by the *client* to a file system. A client *mounts* one of the volumes of the MRC in a local directory. It translates file system calls into RPCs sent to the respective servers.

The 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.

Security

As usual, XtreemFS security differentiates between authentication and authorization. *Authentication* is the process of verifying a user's or client's identity, e.g. validating and reading an X.509 certificate. In contrast, *authorization* is the process of checking if a user has the permission to execute a certain operation, e.g. write access to a file.

By default, XtreemFS uses unauthenticated and unencrypted TCP connections. However, *SSL* can be enabled in all XtreemFS services and the client. Using SSL requires that all users and services provide valid X.509 certificates. Any data sent over a SSL connection is encrypted. Using SSL, however, will increase the resource consumption of all components, especially for connection setup (SSL handshake).

2.2.3 Policies

Many facets of the behavior of XtreemFS can be configured by means of policies. A policy defines how a certain task is performed, e.g. how the MRC selects a set of OSDs for a new file, or how it distinguishes between an authorized and an unauthorized user when files are accessed. Various policies have been defined that cover different aspects.

OSD Selection Policies

When a **new file is created**, the MRC must decide which OSDs to use for storing the file content. Based on the required number of OSDs defined in the file's striping policy, an OSD Selection Policy is responsible for selecting the most suitable OSDs. OSD selection policies are assigned at volume granularity. Currently, there are the following policies:

- Random OSD Selection (policy id 1)
  Randomly selects OSDs from the list of all available OSDs that are alive and have more than 2GB of free space left.

- Proximity-based OSD Selection (policy id 2)
  Selects a group of OSDs that are close to each other. The distance is determined by the IP address, i.e. OSDs on the same subnet are preferred. This policy is particularly useful for striping, since it is desirable to have all OSDs at the same site.

Striping Policies

XtreemFS allows the content of a file to be distributed among several storage devices (OSDs). This has the benefit, that the file can be read or written in parallel on multiple servers which increases the bandwidth. The more OSDs are used, the higher the bandwidth available for reading or writing. The number of OSDs is called the striping width.

A striping policy is a rule that defines how the objects are distributed on the available OSDs. Currently, XtreemFS implements only the `RAID0` policy which simply stores the objects in a round robin fashion on the OSDs. The `RAID0` policy has two parameters. The striping width defines to how many OSDs the file is distributed. The stripe size defines the size of each object.

When using a striping width of 1, the files are not striped but each file is stored on a single OSD. In that case, you can use any OSD Selection Policy which suits your needs. For striped files (i.e. a striping width larger than 1) we recommend to use the Proximity-based OSD selection policy, because the OSDs onto the files are striped should reside on the same network for better performance and data availability.

Striping over several OSDs enhances the read and write bandwidth of a file, the bandwidth increases the larger the striping width. Please note, that striping also increases the probability of data loss. A striped file will become corrupted even if a single OSDs it is stored on has a disk crash.

Authorization - Access Policies

User authorization is managed by means of Access Policies. An access policy defines the access rights for any user on any file or directory contained in a volume. When creating a new volume, the access policy has to be chosen, which cannot be changed in the future. Various access policies can be used:

- Authorize All Policy (policy Id 1)
  No authorization - everyone can do everything. This policy is useful if performance of metadata operations matters more than security, since no recursive evaluation of access policies is done.

- POSIX ACLs & Permissions (policy Id 2)
  This access policy implements the traditional POSIX permissions commonly used on Linux, as well as POSIX ACLs, an extension that provides for access control at the granularity of single users and groups. POSIX permissions should be used as the default, as it guarantees maximum compatibility with other file systems.

- Volume ACLs (policy Id 3)
  Volume ACLs provide an access control model similar to POSIX ACLs

& Permissions, but only allow one ACL for the whole volume. This means that there is no recursive evaluation of access rights which yields a higher performance at the price of a very coarse-grained access control.

Pluggable Policies

Administrators may extend the set of existing policies by defining *plug-in policies*. Such policies are Java classes that implement a predefined policy interface. Currently, the following policy interfaces exist:

- `org.xtreemfs.common.auth.AuthenticationProvider`
  can be used to implement an individual mechanism to authenticate users and groups
- `org.xtreemfs.mrc.ac.FileAccessPolicy`
  can be used to implement an individual access control model on files, directories and volumes
- `org.xtreemfs.mrc.osdselection.OSDSelectionPolicy`
  can be used to implement an individual policy for allocating OSDs to newly created files

Note that there may only be one authentication provider per MRC, while file access policies and OSD selection policies may differ for each volume. The former one is identified by means of its class name (property `authentication_provider`, see Sec. 2.3.2), while volume-related policies are identified by ID numbers. It is therefore necessary to add a member field

```
public static final long POLICY_ID = 4711;
```

to all such policy implementations, where `4711` represents the individual ID number. Administrators have to ensure that such ID numbers neither clash with ID numbers of built-in policies (1-9), nor with ID numbers of other plug-in policies. When creating a new volume, IDs of plug-in policies may be used just like built-in policy IDs.

Plug-in policies have to be deployed in the directory specified by the MRC configuration property `policy_dir`. The property is optional; it may be omitted if no plug-in policies are supposed to be used. An implementation of a plug-in policy can be deployed as a Java source or class file located in a directory that corresponds to the package of the class. Library dependencies may be added in the form of source, class or JAR files. JAR files have to be deployed in the top-level directory. All source files in all subdirectories are compiled at MRC start-up time and loaded on demand.

2.3 XtreemFS Services

2.3.1 Installation

When installing XtreemFS, you can choose from two different installation sources: you can download one of the *pre-packaged releases* that we create for most Linux distributions or you can install directly from the *source tarball*. In the pre-packaged release, the server and the client parts are split into separate packages.

Prerequisites

For the pre-packaged release, you will need Sun Java JRE 1.6.0 or newer to be installed on the system.

When building XtreemFS directly from the source, you need a Sun Java JDK 1.6.0 or newer, Ant 1.6.5 or newer and gmake.

Installing from Pre-Packaged Releases

On RPM-based distributions (RedHat, Fedora, SuSE, Mandriva, XtreemOS) you can install the package with

```
$> rpm -i xtreemfs-server-0.10.x.rpm
```

For Debian-based distributions, please use the `.deb` package provided and install it with

```
$> dpkg -i xtreemfs-server-0.10.x.deb
```

Both packages will also install `init.d` scripts for an automatic start-up of the services. Use `insserv xtreemfs-dir`, `insserv xtreemfs-mrc` and `insserv xtreemfs-osd`, respectively, to automatically start the services during boot.

Installing from Sources

Extract the tarball with the sources. Change to the top level directory and execute

```
$> make server
```

After successful build, you can use the provided installer

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

The installer script will guide you through a basic setup, install the services and prepare start and stop scripts.

2.3.2 Configuration

Generally, the configuration files of XtreemFS are located in `/etc/xos/xtreemfs/` if you installed from packages. If you used the installer for the source distribution, the configuration file can be found in `<INSTALLDIR>/config/` where `<INSTALLDIR>` refers to the XtreemFS installation directory that you have chosen during installation.

A Word about UUIDs

XtreemFS uses UUIDs (Universally Unique Identifiers) to be able to identify services and their associated state independently from the machine they are installed on. This implies that you cannot change the `uuid` of a MRC or OSD after it has been used for the first time!

The Directory Service keeps a mapping from UUID to a port number and IP address or hostname. Currently, each UUID can only be associated with a single endpoint; the netmask must be "*" which means that this mapping is valid in all networks. Upon first start-up, OSDs and MRCs will create the mapping if it does not exist. They will use the first available network device with a public address.

Changing the IP address, hostname or port is possible at any time. Due to the caching of UUIDs in all components it can take some time until the new UUID mapping is used by all OSDs, MRCs and clients. The TTL defines how long an XtreemFS component is allowed to keep entries cached. The default value is 3600 seconds (1 hour). It should be set to shorter durations if services change their IP address frequently.

To create a globally unique UUID you can use tools like `uuidgen`. During installation the post-install script will automatically create a UUID for each OSD and MRC if it does not have a UUID assigned.

Authentication

XtreemFS has an interface which allows MRC administrators to choose the way of authenticating users. Basically, an MRC has two sources of information on users. The first one is the user id and group ids sent by the client along with each request. In addition, the MRC can use information included in the certificates if SSL is enabled. The Authentication Providers are modules that implement different methods for retrieving the user and group IDs to use.

UNIX uid/gid - NullAuthProvider The NullAuthProvider is the default Authentication Provider. It simply uses the user ID and group IDs sent by the XtreemFS client. This means that the client is trusted to send the correct user/group IDs.

The XtreemFS Client will send the user ID and group IDs of the process which executed the file system operation, not of the user who mounted the volume!

The superuser is identified by the user ID `root` and is allowed to do everything on the MRC. This behavior is similar to NFS with `no_root_squash`.

Plain SSL Certificates - SimpleX509AuthProvider XtreemFS supports two X.509 certificate "types" which can be used by the client. When mounted with a service/host certificate the XtreemFS client is regarded as a trusted system component. The MRC will accept any user ID and groups sent by the client and use them for authorization as with the NullAuthProvider. This setup is useful for volumes which are used by multiple users.

The second certificate type are regular user certificates. The MRC will only accept the user name and group from the certificate and ignore the user ID and groups sent by the client. Such a setup is useful if users are allowed to mount XtreemFS from untrusted machines.

Both certificates are regular X.509 certificates. Service and host certificates are identified by a Common Name (`CN`) starting with `host/` or `xtreemfs-service/`, which can easily be used in existing security infrastructures. All other certificates are assumed to be user certificates.

If a user certificate is used, XtreemFS will take the Distinguished Name (`DN`) as the user ID and the Organizational Unit (`OU`) as the group ID.

Superusers must have `xtreemfs-admin` as part of their Organisational Unit (`OU`).

XtreemOS Certificates - XOSAuthProvider In contrast to plain X.509 certificates, XtreemOS embeds additional user information as extensions in XtreemOS-User-Certificates. This authentication provider uses this information (gloabl UID and global GIDs), but the behaviour is similar to the SimpleX509AuthProvider.

The superuser is identified by being member of the `VOAdmin` group.

List of Configuration Options

All configuration parameters that may be used to define the behavior of the different services are listed in the following. Unless marked as optional, a parameter has to occur (exactly once) in a configuration file.

`authentication_provider`

Services	DIR, MRC
Values	Java class name
Default	`org.xtreemfs.common.auth.NullAuthProvider`
Description	Defines the Authentication Provider to use to retrieve the user identity (user ID and group IDs). See Sec. 2.3.2 for details.

`capability_secret`

Services	MRC, OSD
Values	String
Default	-
Description	Defines a shared secret between the MRC and all OSDs. The secret is used by the MRC to sign capabilities, i.e. security tokens for data access at OSDs. In turn, an OSD uses the secret to verify that the capability has been issued by the MRC.

`checksums.enabled`

Services	OSD
Values	true, false
Default	false
Description	If set to true, the OSD will calculate and store checksums for newly created objects. Each time a checksummed object is read, the checksum will be verified.

`checksums.algorithm`

Services	OSD
Values	Adler32, CRC32, MD5, SHA-1
Default	Adler32
Description	Must be specified if `checksums.enabled` is enabled. This property defines the algorithm used to create OSD checksums.

`database.dir`

Services	DIR, MRC
Values	absolute file system path to a directory
Default	DIR: `/var/lib/xtreemfs/dir/database`, MRC: `/var/lib/xtreemfs/mrc/database`
Description	The directory in which the Directory Service or MRC will store their databases. This directory should never be on the same partition as any OSD data, if both services reside on the same machine. Otherwise, deadlocks may occur if the partition runs out of free disk space!

`database.log`

Services	MRC
Values	absolute file system path
Default	MRC: `/var/lib/xtreemfs/mrc/dblog`
Description	The file the MRC uses as the database operations log. This directory should never be on the same partition as any OSD data, if both services reside on the same machine. Otherwise, deadlocks may occur if the partition runs out of free disk space!

`database.checkpoint.interval`

Services	MRC
Values	milliseconds
Default	180,000
Description	The MRC regularly checks if it is necessary to create a database checkpoint on disk. This parameter specifies the interval between two checks.

`database.checkpoint.idle_interval`

Services	MRC
Values	milliseconds
Default	1,000
Description	The MRC will only create a checkpoint if there have been no client requests for the last `database.checkpoint.idle_interval` milliseconds.

`database.checkpoint.logfile_size`

Services	MRC
Values	kilobytes
Default	16,384
Description	A checkpoint of the database will only be created if the `database.log` has exceeded the specified file size.

`debug_level`

Services	DIR, MRC, OSD
Values	0, 1, 2, 3, 10
Default	2
Description	The debug level determines the amount and detail of information written to logfiles. 0 logs errors only, 1 logs additional warnings, 2 logs errors, warnings and info, 3 logs errors, warnings, info and debug messages (generats large logfiles!), 10 also catches tracing output (generates very large logfiles!).

`dir_service.host`

Services	MRC, OSD
Values	hostname or IP address
Default	localhost
Description	Specifies the hostname or IP address of the directory service (DIR) at which the MRC or OSD should register. The MRC also uses this directory service to find OSDs.

`dir_service.port`

Services	MRC, OSD
Values	1 .. 65535
Default	32638
Description	Specifies the port on which the remote directory service is listening. Must be identical to the `listen_port` in your directory service configuration.

`listen.address` ***optional***

Services	OSD
Values	IP address
Default	-
Description	If specified, defines the interface to listen on. If not specified, the service will listen on all interfaces (any).

`listen.port`

Services	DIR, MRC, OSD
Values	1 .. 65535
Default	DIR: 32638, MRC: 32636, OSD: 32640
Description	The port to listen on for incoming connections (TCP). The OSD uses TCP and UDP on the specified port. Make sure to configure your firewall to allow incoming TCP and UDP traffic on the specified port.

`local_clock_renewal`

Services	MRC, OSD
Values	milliseconds
Default	50
Description	Reading the system clock is a slow operation on some systems (e.g. Linux) as it is a system call. To increase performance, XtreemFS services use a local variable which is only updated every `local_clock_renewal` milliseconds.

`no_atime`

Services	MRC
Values	true, false
Default	true
Description	The POSIX standard defines that the atime (timestamp of last file access) is updated each time a file is opened, even for read. This means that there is a write to the database and hard disk on the MRC each time a file is read. To reduce the load, many file systems (e.g. ext3) including XtreemFS can be configured to skip those updates for performance. It is strongly suggested to disable atime updates by setting this parameter to true.

`no_fsync` ***optional***

Services	MRC
Values	true, false
Default	false
Description	By default, the MRC will write all file-modifying operations (such as create file, delete etc.) to disk followed by a `fsync` to ensure data is written to the hard disk. While this ensures maximum data safety in case of crash of the MRC server, it also reduces the performance of the MRC. Set this to true, if you want much higher performance at the risk of losing some recent file operations in case of a server crash.

`object_dir`

Services	OSD
Values	absolute file system path to a directory
Default	`/var/lib/xtreemfs/osd/`
Description	The directory in which the OSD stores the objects. This directory should never be on the same partition as any DIR or MRC database, if both services reside on the same machine. Otherwise, deadlocks may occur if the partition runs out of free disk space!

`osd_check_interval`

Services	MRC
Values	seconds
Default	300
Description	The MRC regularly asks the directory service for suitable OSDs to store files on (see OSD Selection Policy, Sec. 2.2.3). This parameter defines the interval between two updates of the list of suitable OSDs.

`remote_time_sync`

Services	MRC, OSD
Values	milliseconds
Default	30,000
Description	MRCs and OSDs all synchronize their clocks with the directory service to ensure a loose clock synchronization of all services. This is required for leases to work correctly. This parameter defines the interval in milliseconds between time updates from the directory service.

`report_free_space`

Services	OSD
Values	true, false
Default	true
Description	If set to true, the OSD will report its free space to the directory service. Otherwise, it will report zero, which will cause the OSD not to be used by the OSD Selection Policies (see Sec. 2.2.3).

`ssl.enabled`

Services	DIR, MRC, OSD
Values	true, false
Default	false
Description	If set to true, the service will use SSL to authenticate and encrypt connections. The service will not accept non-SSL connections if `ssl.enabled` is set to true.

`ssl.service_creds`

Services	DIR, MRC, OSD
Values	path to file
Default	DIR: `/etc/xos/xtreemfs/truststore/certs/ds.p12`, MRC: `/etc/xos/xtreemfs/truststore/certs/mrc.p12`, OSD: `/etc/xos/xtreemfs/truststore/certs/osd.p12`
Description	Must be specified if `ssl.enabled` is enabled. Specifies the file containing the service credentials (X.509 certificate and private key). PKCS#12 and JKS format can be used, set `ssl.service_creds.container` accordingly. This file is used during the SSL handshake to authenticate the service.

`ssl.service_creds.container`

Services	DIR, MRC, OSD
Values	`pkcs12` or `JKS`
Default	`pkcs12`
Description	Must be specified if `ssl.enabled` is enabled. Specifies the file format of the `ssl.service_creds` file.

`ssl.service_creds.pw`

Services	DIR, MRC, OSD
Values	String
Default	-
Description	Must be specified if `ssl.enabled` is enabled. Specifies the password which protects the credentials file `ssl.service_creds`.

`ssl.trusted_certs`

Services	DIR, MRC, OSD
Values	path to file
Default	`/etc/xos/xtreemfs/truststore/certs/xosrootca.jks`
Description	Must be specified if `ssl.enabled` is enabled. Specifies the file containing the trusted root certificates (e.g. CA certificates) used to authenticate clients.

`ssl.trusted_certs.container`

Services	DIR, MRC, OSD
Values	`pkcs12` or `JKS`
Default	`JKS`
Description	Must be specified if `ssl.enabled` is enabled. Specifies the file format of the `ssl.trusted_certs` file.

`ssl.trusted_certs.pw`

Services	DIR, MRC, OSD
Values	String
Default	-
Description	Must be specified if `ssl.enabled` is enabled. Specifies the password which protects the trusted certificates file `ssl.trusted_certs`.

`uuid`

Services	MRC, OSD
Values	String, but limited to alphanumeric characters, - and .
Default	-
Description	Must be set to a unique identifier, preferably a UUID according to RFC 4122. UUIDs can be generated with `uuidgen`. Example: `eacb6bab-f444-4ebf-a06a-3f72d7465e40`.

DIR Configuration

The directory service configuration is stored in `dirconfig.properties`.

```
debug_level = 0
listen.port = 32638
database.dir = /var/lib/xtreemfs/dir/database
ssl.enabled = false
authentication_provider = org.xtreemfs.common.auth.NullAuthProvider
ssl.service_creds = /etc/xos/xtreemfs/truststore/certs/ds.p12
ssl.service_creds.pw = xtreemfs
ssl.service_creds.container = pkcs12
ssl.trusted_certs = /etc/xos/xtreemfs/truststore/certs/xosrootca.jks
ssl.trusted_certs.pw = xtreemfs
ssl.trusted_certs.container = jks
```

MRC Configuration

The directory service configuration is stored in `mrcconfig.properties`.

```
debug_level = 0
listen.port = 32636
dir_service.host = localhost
dir_service.port = 32638
database.dir = /var/lib/xtreemfs/mrc/database
database.log = /var/lib/xtreemfs/mrc/dblog
database.checkpoint.interval = 1800000
database.checkpoint.idle_interval = 1000
database.checkpoint.logfile_size = 16384
osd_check_interval = 300
```

```
no_atime = true
local_clock_renewal = 50
remote_time_sync = 60000
uuid = eacb6bab-f444-4ebf-a06a-3f72d7465e40
authentication_provider = org.xtreemfs.common.auth.NullAuthProvider
ssl.enabled = false
ssl.service_creds = /etc/xos/xtreemfs/truststore/certs/ds.p12
ssl.service_creds.pw = xtreemfs
ssl.service_creds.container = pkcs12
ssl.trusted_certs = /etc/xos/xtreemfs/truststore/certs/xosrootca.jks
ssl.trusted_certs.pw = xtreemfs
ssl.trusted_certs.container = jks
```

OSD Configuration

The OSD service configuration is stored in `osdconfig.properties`.

```
debug_level = 0
listen.port = 32640
listen.address = 127.0.0.1
dir_service.host = localhost
dir_service.port = 32638
object_dir = /var/lib/xtreemfs/objs/
local_clock_renewal = 50
remote_time_sync = 60000
report_free_space = true
uuid = eacb6bab-f444-4ebf-a06a-3f72d7465e41
ssl.enabled = false
ssl.service_creds = /etc/xos/xtreemfs/truststore/certs/ds.p12
ssl.service_creds.pw = xtreemfs
ssl.service_creds.container = pkcs12
ssl.trusted_certs = /etc/xos/xtreemfs/truststore/certs/xosrootca.jks
ssl.trusted_certs.pw = xtreemfs
ssl.trusted_certs.container = jks
```

Configuring SSL Support

In order to enable certificate-based authentication in an XtreemFS installation, services need to be equipped with X.509 certificates. Certificates are used to establish a mutual trust relationship among XtreemFS services and between the XtreemFS client and XtreemFS services.

It is not possible to mix SSL-enabled and non-SSL services in an XtreemFS installation!

Each XtreemFS service needs a certificate and a private key in order to be run. Once they have created and signed, the credentials may need to be converted into the correct file format. XtreemFS services also need a *trust store* that contains all trusted Certification Authority certificates.

By default, certificates and credentials for XtreemFS services are stored in

```
/etc/xos/xtreemfs/truststore/certs
```

Converting PEM files to PKCS#12

The simplest way to provide the credentials to the services is by converting your signed certificate and private key into a PKCS#12 file using `openssl`:

```
$> 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. The passwords chosen when asked must be set as a property in the corresponding service configuration file.

Importing trusted certificates from PEM into a JKS

The certificate (or multiple certificates) from your CA (or CAs) can be imported into a Java Keystore (JKS) using the Java keytool which comes with the Java JDK or JRE.

Execute the following steps for each CA certificate using the same keystore file.

```
$> keytool -import -alias rootca -keystore trusted.jks
   -trustcacerts -file ca-cert.pem
```

This will create a new Java Keystore `trusted.jks` with the CA certificate in the current working directory. The password chosen when asked must be set as a property in the service configuration files.

Note: If you get the following error

```
$> keytool error: java.lang.Exception: Input not an X.509 certificate
```

you should remove any text from the beginning of the certificate (until the `-----BEGIN CERTIFICATE-----` line).

Sample Setup

Users can easily set up their own CA (certificate authority) and create and sign certificates using `openssl` for a test setup.

1. Set up your test CA.
   (a) Create a directory for your CA files
   ```
   $> mkdir ca
   ```
   (b) Create a private key and certificate request for your CA.
   ```
   $> openssl req -new -newkey rsa:1024 -nodes -out ca/ca.csr \
      -keyout ca/ca.key
   ```
   Enter something like XtreemFS-DEMO-CA as the common name (or something else, but make sure the name is different from the server and client name!).

   (c) Create a self-signed certificate for your CA which is valid for one year.
   ```
   $> openssl x509 -trustout -signkey ca/ca.key -days 365 -req\
      -in ca/ca.csr -out ca/ca.pem
   ```
   (d) Create a file with the CA's serial number
   ```
   $> echo "02" > ca/ca.srl
   ```
2. Set up the certificates for the services and the XtreemFS Client. Replace *`service`* with `dir`, `mrc`, `osd` and `client`.
   (a) Create a private key for the service.
   Use `XtreemFS-DEMO-`*`service`* as the common name for the certificate.

```
$> openssl req -new -newkey rsa:1024 -nodes
   -out service.req
   -keyout service.key
```

(b) Sign the certificate with your demo CA.
The certificate is valid for one year.

```
$> openssl x509 -CA ca/ca.pem -CAkey ca/ca.key
   -CAserial ca/ca.srl -req
   -in service.req
   -out service.pem -days 365
```

(c) Export the service credentials (certificate and private key) as a PKCS#12 file.
Use "passphrase" as export password. You can leave the export password empty for the XtreemFS Client to avoid being asked for the password on mount.

```
$> pkcs12 -export -in service.pem -inkey
   service.key
   -out service.p12 -name "service"
```

(d) Copy the PKCS#12 file to the certificates directory.

```
        $> mkdir -p /etc/xos/xtreemfs/truststore/certs
    $> cp service.p12 /etc/xos/xtreemfs/truststore/certs
```

3. Export your CA's certificate to the trust store and copy it to the certificate dir.
You should answer "yes" when asked "Trust this certificate".
Use "passphrase" as passphrase for the keystore.

```
$> keytool -import -alias ca -keystore trusted.jks\
   -trustcacerts -file ca/ca.pem
$> cp  trusted.jks /etc/xos/xtreemfs/truststore/certs
```

4. Configure the services. Edit the configuration file for all your services.
Set the following configuration options (see Sec. 2.3.2 for details).

```
use_ssl = true
ssl_service_creds_pw = passphrase
ssl_service_creds_container = pkcs12
ssl_service_creds = /etc/xos/xtreemfs/truststore/certs/service.p12
ssl_trusted_certs = /etc/xos/xtreemfs/truststore/certs/trusted.jks
ssl_trusted_certs_pw = passphrase
ssl_trusted_certs_container = jks
```

5. Start up the XtreemFS services (see Sec. 2.3.3).
6. Create a new volume (see Sec. 2.3.3 for details).

   ```
   $> xtfs_mkvol -c /etc/xos/xtreemfs/truststore/certs/client.p12 \
      -p RAID0,256,1 https://localhost/test
   ```

7. Mount the volume (see Sec. 2.4.2 for details).

   ```
   $> xtfs_mount -o ssl_cert=\
      /etc/xos/xtreemfs/truststore/certs/client.p12, \
      dirservice=https://localhost, \
      volume_url=https://localhost/test /mnt
   ```

2.3.3 Management

This section covers all tools and functionality for XtreemFS management and tracing. In general, the use of management tools is restricted to superusers.

Starting and Stopping the XtreemFS services

If you installed a *pre-packaged release* you can start, stop and restart the services with the `init.d` scripts:

```
$> /etc/init.d/xtreemfs-ds start
$> /etc/init.d/xtreemfs-mrc start
$> /etc/init.d/xtreemfs-osd start
```

or

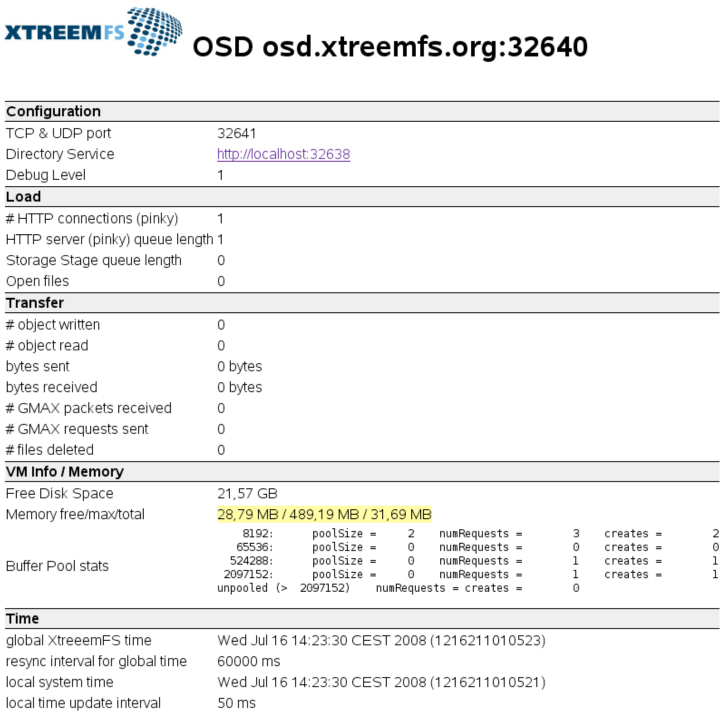
```
$> /etc/init.d/xtreemfs-ds stop
$> /etc/init.d/xtreemfs-mrc stop
$> /etc/init.d/xtreemfs-osd stop
```

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.

If you installed from sources, you will find a `start.sh` and `stop.sh` script in the install directory. These scripts will automatically start/stop all services you installed on the machine.

Web-based Status Page

The XtreemFS services all have a HTML status page which can be used to check if the service is working correctly (Fig. 2.2). It can be displayed by opening the service URL in your favorite web browser, e.g. `http://my-mrc-host.com:32636/`. If you use SSL you should first import the client credentials (PKCS#12 file) into your webbrowser's credential store.



XTREEMFS OSD osd.xtreemfs.org:32640

Configuration	
TCP & UDP port	32641
Directory Service	http://localhost:32638
Debug Level	1
Load	
# HTTP connections (pinky)	1
HTTP server (pinky) queue length	1
Storage Stage queue length	0
Open files	0
Transfer	
# object written	0
# object read	0
bytes sent	0 bytes
bytes received	0 bytes
# GMAX packets received	0
# GMAX requests sent	0
# files deleted	0
VM Info / Memory	
Free Disk Space	21,57 GB
Memory free/max/total	28,79 MB / 489,19 MB / 31,69 MB

Buffer Pool stats	`8192: poolSize = 2 numRequests = 3 creates = 2`
	`65536: poolSize = 0 numRequests = 0 creates = 0`
	`524288: poolSize = 0 numRequests = 1 creates = 1`
	`2097152: poolSize = 0 numRequests = 1 creates = 1`
	`unpooled (> 2097152) numRequests = creates = 0`

Time	
global XtreeemFS time	Wed Jul 16 14:23:30 CEST 2008 (1216211010523)
resync interval for global time	60000 ms
local system time	Wed Jul 16 14:23:30 CEST 2008 (1216211010521)
local time update interval	50 ms

Figure 2.2: OSD status web page

Creating Volumes

Volumes can be created with the `xtfs_mkvol` command line utility. Please see `man xtfs_mkvol` for a full list of options and usage.

When creating a volume, it is recommended to specify the access policy (see Sec. 2.2.3). If not specified, POSIX permissions/ACLs will be chosen by default. Access policies cannot be changed afterwards.

An OSD selection policy (see Sec. 2.2.3) can also be specified per volume, but can be changed anytime. By default, a random selection of available OSDs is assigned to newly created files.

In addition, it is recommended to set a default striping policy (see Sec. 2.2.3). If no per-file or per-directory default striping policy overrides the volume's default striping policy, the volume's policy is used for new files and directories. If no volume policy is explicitly defined, a RAID0 policy with a stripe size of 4kB and a width of 1 will be assigned to the volume.

An example call to `xtfs_mkvol` for creating a volume with POSIX ACLs, 256kB stripe size and a stripe width of 1 (which means no striping):

```
$> xtfs_mkvol -a 2 -p RAID0,256,1 \
   http://my-mrc-host.com:32636/myVolume
```

Currently, there are no restrictions on the creation of volumes; any user may create a new volume.

Deleting Volumes

The `xtfs_rmvol` tool can be used to delete a volume. This also *deletes all files and data on that volume*! Please see `man xtfs_rmvol` for a full list of options and usage.

Example call to `xtfs_rmvol` to delete `myVolume`:

```
$> xtfs_rmvol http://my-mrc-host.com:32636/myVolume
```

Volume deletion is restricted to volume owners and superusers.

MRC Database Conversion

The format in which the MRC stores its data on disk may change with future XtreemFS versions. In order that XtreemFS server components may be updated without losing the whole content of the file system, it is possible to create a version-independent XML representation of the metadata stored in MRC database.

Such an XML representation can e.g. be created as follows:

```
$> xtfs_mrcdbtool http://my-mrc-host.com:32636 \
   dump /tmp/dump.xml
```

This call will create a file `dump.xml` containing the entire MRC database content in the `/tmp` directory at `my-mrc-host.com`.

To restore an MRC database from a dump, execute

```
$> xtfs_mrcdbtool http://my-mrc-host.com:32636 \
   restore /tmp/dump.xml
```

This will restore the database stored in `/tmp/dump.xml` at `my-mrc-host.com`. Note that for safety reasons, it is only possible to restore a database from a dump if the database of the running MRC does not have any content. To restore an MRC database, it is thus necessary to delete all MRC database files before starting the MRC.

Scrubbing and Cleanup

In real-world environments, errors occur in the course of creating, modifying or deleting files. This can cause corruptions of file data or metadata. Such things happen e.g. if the client is suddenly terminated, or loses connection with a server component. There are several such scenarios: if a client writes to a file but does not report file sizes received from the OSD back to the MRC, inconsistencies between the file size stored in the MRC and the actual size of all objects in the OSD will occur. If a client deletes a file from the directory tree, but cannot reach the OSD, orphaned objects will remain on the OSD. If an OSD is terminated during an ongoing write operation, file content will become corrupted.

In order to detect and, if possible, resolve such inconsistencies, tools for scrubbing and OSD cleanup exist. To check the consistency of file sizes and checksums, the following command can be executed:

```
$> xtfs_scrub -dir http://my-dir-host.com:32638 \
   http://my-mrc-host.com:32636/myVolume
```

This will scrub each file in the volume `myVolume`, i.e. check file size consistency and set the correct the file size on the MRC, if necessary, and check whether an invalid checksum in the OSD indicates a corrupted file content. The `-dir` argument specifies the directory service that will be used to resolve service UUIDs. Please see `man xtfs_scrub` for further details.

A second tool searches an OSD for orphaned objects, which can be used as follows:

```
$> xtfs_cleanup -dir http://my-dir-host.com:32638 \
   http://my-osd-host.com:32640
```

This will touch all objects stored on the given OSD and check whether a metadata representation exists on the responsible MRC. If this is not the case, the objects may either be deleted, or assigned to new files in a `lost+found` directory at the top level of the volume. Please see `man xtfs_cleanup` for further details.

2.4 The XtreemFS Client

2.4.1 Installation

As for the XtreemFS Services, there are two different installation sources for the XtreemFS Client: *pre-packaged releases* and *source tarballs*.

Prerequisites

For both installations you need FUSE 2.6 or newer, openSSL 0.9.8 or newer and a Linux 2.6 kernel.

To build the XtreemFS Client from sources, you need the openSSL headers (e.g. openssl-devel package), gmake 3.81 or newer and gcc 4.1.2 or newer.

Installing from Pre-Packaged Releases

On RPM-based distributions (RedHat, Fedora, SuSE, Mandriva, XtreemOS) you can install the package with

```
$> rpm -i xtreemfs-client-0.10.x.rpm
```

For Debian-based distributions, please use the `.deb` package provided and install it with

```
$> dpkg -i xtreemfs-client-0.10.x.deb
```

Installing from Sources

Extract the tarball with the sources. Change to the top level directory and execute

```
$> make client
```

You can copy the driver `xtfs_mount` from `AL/src` and the utilities from the `AL/tools` directory.

2.4.2 Mounting and Un-mounting

Before mounting XtreemFS volumes, please ensure that the FUSE kernel module is loaded. Please check your distribution's manual to see, if users must be in a special group (e.g. `trusted` in openSUSE) to be allowed to mount FUSE.

```
$> su
Password:
#> modprobe fuse
#> exit
```

To mount an XtreemFS volume use the `xtfs_mount` tool.

```
$> xtfs_mount -o dirservice=http://remote.dir.machine \
   -o volume_url=http://remote.mrc.machine/myVolume \
   /xtreemfs
```

The `-o volume_url=`URL option is mandatory and specifies which volume to mount. `-o dirservice=`URL option is mandatory as well and must point to the directory service. The first and only argument points to a directory on the local file system in which to mount the XtreemFS volume. For more options, please refer to `man xtfs_mount`.

A fuse mount is normally private for one user. To allow other users on the system or permit root to use the mounted volume as well, the `-o allow_other` and `-o allow_root` options can be passed to `xtfs_mount`, respectively. They are, however, mutually exclusive. In order to use these options, the system administrator must create a FUSE configuration file `/etc/fuse.conf` and add a line `user_allow_other`.

Volumes are unmounted using the `xtfs_umount` tool.

```
$> xtfs_umount /xtreemfs
```

2.4.3 Reading XtreemFS-specific File Info

In addition to the regular file system information provided by the `stat` Linux utility, XtreemFS provides the `xtfs_stat` tool which displays XtreemFS specific information for a file or directory.

```
$> cd /xtreemfs
$> echo 'Hello World' > test.txt
$> xtfs_stat test.txt
```

will produce output similar to the following

```
filename            test.txt
XtreemFS URI        uuid:eacb6bab-f444-4ebf-a06a-3f72d7465e40/xtreemfs/test.txt
XtreemFS fileID     0004760EDB989891BD4774E2:398564
object type         regular file
owner               xtreemfs
group               users
read-only           false
replica list ver.   1
replica             1 ( policy: RAID0, width: 2, stripe-size: 512kB )
      OSD 1         eacb6bab-f444-4ebf-a06a-3f72d7465e41
      OSD 2         9aea34d3-bce4-4948-b4ff-0af31f3fd229
```

The XtreemFS url can be used to retrieve the volume URL and the path of the file on the volume. The fileID is the unique identifier of the file used on the OSDs to identify the file's objects. The owner/group fields are shown as reported by the MRC, you may see other names on your local system if there is no mapping (i.e. the file owner does not exist as a user on your local machine). Finally, the XtreemFS replica list shows the striping policy of the file, the number of replicas and for each replica, the OSDs used to store the objects.

2.4.4 Changing Striping Policies

It is not (yet) possible to change the striping policy of an existing file, as this would require moving and reformatting data among OSDs. However, individual striping policies can be assigned to new files (i.e. empty files) by changing the default striping policy of the parent directory or volume. For this purpose, XtreemFS provides the `xtfs_sp` tool. The tool can be used to change the striping policy that will be assigned to newly created files.

```
$> xtfs_sp set /xtreemfs RAID0,128,3
```

In addition, the tool can be used to retrieve the default striping policy of a volume or directory.

```
$> xtfs_sp get /xtreemfs
```

The output will be similar to the following:

```
RAID0,128,3
```

When creating a new file, XtreemFS will first check whether a default striping policy has been assigned to the parent directory. If this is not the case, the default striping policy for the volume will be used as the striping policy for the new file. Changing a volume's or directory's default striping policy requires superuser access rights or ownership of the directory or volume.

2.5 Troubleshooting and Support

2.5.1 Logfiles

The logfiles for the XtreemFS services are located in `/var/log/xtreemfs`. The client logfile must be specified with the `-o logfile=/var/log/xtreemfs/client1.log` mount option to `xtfs_mount`, otherwise the client messages will go to your syslog.

2.5.2 Support

Please visit the XtreemFS website at www.XtreemFS.org for links to the user mailing list and IRC channel.

2.5.3 Troubleshooting

Problem: The client hangs when opening/copying/creating a file but operations like `ls` or `mkdir` work.

Solution: This problem can occurr when an OSD uses a UUID which resolves to an address that the client cannot (correctly) resolve. For instance, if you use a UUID for the OSD that resolves to `http://localhost:32640`, the client will try to contact the local machine instead of the machine on which the OSD runs. Open the status page of your Directory Service and check the UUID of the OSDs.

Chapter 3

The OSS API and User Guide

3.1 What is OSS

The Object Sharing Service (OSS) implements distributed objects for nodes participating in an interactive multi-user grid application. OSS runs on each client machine to enable sharing of objects residing in volatile memory. An object in this context is a replicated volatile memory region, dynamically allocated by an application or mapped into memory from a file.

Objects may contain scalars, references, and code. Therefore, OSS handles concurrent read and write access to objects and maintains the consistency of replicated objects. Persistence and security for objects stored in files are provided by XtreemFS. Fault tolerance is provided by the grid checkpointing mechanisms developed in WP3.3. OSS is being developed for Linux on IA32 or AMD64 compatible processors.

3.2 Installation of OSS

You can install OSS either using the prebuild distribution packages, which are for example available on the XtreemOS release media, or you can build and install OSS from source code. We suggest using the first method mentioned, unless you wish to configure special build-time settings for OSS.

3.2.1 Installing OSS Using the Distribution Packages

The XtreemOS release contains the OSS library, as well as a raytracing demo application to demonstrate object sharing. During the XtreemOS installation procedure, simply select the checkbox *Object Sharing Service* to install the packaged version of OSS. If you have XtreemOS already installed and wish to install OSS, select it in the package management dialog, or run the following command as root:

```
$> urpmi liboss-0.2
```

If you are running a Debian-based distribution, please download the `liboss-0.2.deb` package and install it by running the following command as root:

```
$> dpkg -i liboss-0.2.deb
```

3.2.2 Building and Installing OSS from Source

By building and installing OSS from source, you have full control over the installation process. You can configure how OSS is installed, and fine-tune all OSS features. However, a few additional tools and libraries are needed to build OSS.

Prerequisites

If you wish to build and install OSS from the source code, you need to have some additional development packages installed on your build system. The names of these packages depend on which Linux distribution you are using. Although the OSS build process includes diverse checks for these libraries, we cannot anticipate the requirements for building OSS on all Linux distributions available. If in doubt, please consult the package search of your distribution.

Under Debian GNU/Linux, the following packages are needed to build OSS and the included demo applications:

- gcc $\geq$ 4.3
- binutils $\geq$ 2.18
- make

- libc6-dev
- libglib2.0-dev $\geq$ 2.14
- libreadline5-dev

In case your distribution does not include a recent GLib, the OSS installation process helps you download and install GLib from source. Simply run the following command, entering the root password when asked for:

```
$> make build/glib
```

For 32-bit OSS under Linux x86_64 you also need

- ia32-libs
- ia32-libs-gtk

To compile 32-bit OSS under Linux x86_64, some distributions lack symbolic links to 32-bit libraries (libgthread-2.0.so and libglib-2.0.so). When encountering error messages telling that libgthread or glib cannot be found, log in as user root and create the missing symlinks:

```
$> ln -s libgthread-2.0.so.0 /emul/ia32-linux/usr/lib/libgthread-2.0.so
$> ln -s libglib-2.0.so.0 /emul/ia32-linux/usr/lib/libglib-2.0.so
```

The following packages are useful to generate documentation:

- doxygen
- graphviz
- texlive

Doxygen generates source code documentation, whereas graphviz and texlive enable dependency graph and PDF file output respectively.

Compilation

Unpack the OSS source code archive, change to the base directory that just has been created, and type the following command to build the OSS library in the standard configuration:

```
$> make
```

The make system autodetects most tools used for building OSS. If you encounter any errors, please ensure you have a recent compiler and linker installed, and that all developer packages mentioned above are installed correctly.

If you wish to enable non-standard features, you can supply the corresponding parameters to make. For example, run `make -B ARCH=I686` to configure building for 32-Bit x86 machines.

If you prefer running OSS using transactional consistency model with more than two nodes, you must specify the number of participating nodes during compile time. For example, run `make -B CFLAGS+=-DTC_MAX_NODES=<x>`, where x stands for the number of nodes.

In release 0.3 of OSS, the number of nodes will be variable, and a menu-based configuration utility will be included, which is invoked via the command:

```
$> make configure
```

Installation

The following command installs the OSS library on the system, by default in the `/usr/local` hierarchy. For write access to system directories, you need root privileges.

```
$> make install
```

You can change the default installation hierarchy by specifying `prefix=<pathname>`, e.g. to install OSS below `/usr`, run the command

```
$> make install prefix=/usr
```

Software distributors can specify an additional prefix for the actual installation directory by defining `DESTDIR=<additional-prefix>` on the make command line.

3.2.3 Testing the OSS Installation

The OSS make system includes a command to verify that OSS has been installed correctly, and that everything needed for running a program that uses OSS is set up correctly:

```
$> make verify-install
```

The program should output the version and build information of the OSS library found:

```
Object Sharing Service version 0.2 architecture I686
   subversion revision 1297 (2008-09-11 14:37:52)
build 1
Object Sharing Service has been installed correctly.
```

Simple test of Object Sharing

The simple test of object sharing starts two instances of a program: the first process creates a shared object and writes the string "‘hello”’ to it. The second process waits until the object has been created, and as soon as it reads the expected string, it overwrites it with the string "‘world”’.

The Raytracer Application

The raytracer is based on a application developed for a course at the MIT and has been ported to OSS with the focus on testing and demonstrating transactional shared memory. All graphical objects and the image file are allocated in transactional shared memory. Start the first node with

```
$> oss_raytracer --address <IP1>
```

and the subsequent nodes with

```
$> oss_raytracer --address <IPn> --bootstrap <IP1>
```

where `IP1` is the IP address for the first node, and `IPn` is replaced by the IP address of the respective node. To configure the tracing progress, the first node will ask some parameters:

1. Consistency model: ’t’ for transactional consistency, ’s’ for strong consistency
2. Number of Nodes
3. Number of accesses (applies to transactional consistency only): number of accesses between bot and eot
4. Pattern: specify one of ’l’, ’c’, ’p’, ’x’ or ’m’. ’l’ for line by line, ’c’ for column by column, ’p’ for x partitions, ’x’ for every Xth dot, ’m’ for matching pages.
5. Scene: 1, 2, or 3
6. Columns (e.g. 640)
7. Rows (e.g. 480)

After rendering is done, you can give new parameters and render another scene. There are three predefined scenes in this project. Scene1 very simple with one sphere in center and a few bowls around and only a few lights, Scene2 very complex with some arrangements of bowls and reflecting walls. Scene3 just for amusing OSS written with bowls. You can write own scenes as C files analoguous to SceneDemo1.c.

We are currently extending the raytracer with a graphical user interface (GUI), to visualize parallel rendering.

3.3 Developing Applications Using OSS

The internal interface of the OSS library is implementation-dependent, and may be extended in the future, based on insights gained during the development of OSS-based applications. In constrast, WP3.1 is currently defining an XOSAGA interface for object sharing, which will eventually represent the OSS interface in a portable way.

3.3.1 Internal Interface of the OSS Library

The interface of the OSS library is declared in the header file `oss.h`. The following command generates an interface documentation in HTML and (if latex is available) in PDF format:

```
$> make interface-doc
```

The documentation is stored in the `build/doc/` subdirectory.

Let us quickly walk through the basic functionality of the OSS library. For a more detailed and precise discussion of the internal library interface, please see the Doxygen documentation generated directly from the source code. To get a deeper understanding of how to design applications that access shared objects, we suggest looking at the source code examples in the `src/apps/` subdirectory.

```
int
oss_startup(
     const char *addr,
     const char *listen_port,
     const char *bootstrap_addr,
     const char *bootstrap_port
     );
```

The `oss_startup` call starts the OSS system by joining a bootstrap peer. The `addr` and `listen_port` parameters allow to bind the OSS instance to a specific interface. If no bootstrap peer is specified (i.e. a NULL pointer is passed), a new distributed object storage is created. A return value of zero indicates successful startup.

```
void *
oss_alloc(
     size_t size,
     oss_consistency_model_t consistency_model,
     oss_alloc_attributes_t *attributes
     );
```

The `oss_alloc` call creates a shared object of specified size, initializes it with a consistency model and further attributes (defined by the consistency model), and returns an identifier for the object.

```
void
oss_free(
     void *ptr
     );
```

The `oss_free` call frees some memory which has previously been dynamically allocated using `oss_alloc`.

```
oss_transaction_id_t
oss_bot(
     oss_transaction_priority_t priority,
     oss_transaction_attributes_t *attributes
     );
```

The oss_bot call marks the begin of a transaction with given priority and attributes. OSS guarantees that all accesses to distributed objects between oss_bot and oss_eot perform atomically, consistent, isolated, and durable. The return value references the transaction that has been started, or equals oss_undefined_transaction_id which indicates that the transaction failed to start.

```
int
oss_eot(
     oss_transaction_id_t taid
     );
```

The oss_eot call denotes the end of the supplied transaction.

```
int
oss_abort(
     oss_transaction_id_t taid
     );
oss_permit_abort(
     oss_transaction_id_t taid
     );
```

Both calls handle voluntarily aborting a transaction. An application that somehow finds out that it cannot commit, or that committing will have adverse effect, may call oss_abort to unconditionally abort the supplied transaction. Depending on the transaction attributes used, the transaction will restart or simply fail. An application may optionally call oss_permit_abort to mark locations in the code where it is safe to abort a transaction. If the transaction is already known to fail on commit, OSS can restart the transaction and need not delay restarting the transaction until oss_eot. If the success of the transaction is not yet determined, the call to oss_permit_abort will simply appear as a void statement.

3.3.2 Linking Against the OSS Library

The OSS library is built as a static shared library (`liboss.a`) and as a dynamic shared library (`liboss.so`). Simply specify the option `-loss` to the compiler driver or linker, which will link against the appropriate static or dynamic library. If you did not install the library into a well-known location such as `/usr/lib`, you will need to specify the path to the library via the option `-L<path>`.

Appendix A

XtreemOS Integration

XtreemFS Security Preparations

XtreemFS can be integrated in an existing XtreemOS VO security infrastructure. XtreemOS uses X.509 certificates to authenticate users in a Grid system, so the general setup is similar to a normal SSL-based configuration.

Thus, in an XtreemOS environment, certificates have to be created for the services as a first step. This is done by issuing a *Certificate Signing Request (CSR)* to the RCA server by means of the `create-server-csr` command. For further details, see the Section Using the RCA in the XtreemOS User Guide.

Signed certificates and keys generated by are RCA infrastructure are stored locally in PEM format. Since XtreemFS services are currently not capable of processing PEM certificates, 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/xtreemosrootcacert.pem
```

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

```
# specify whether SSL is required
use_ssl = 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. Make sure that all paths and passphrases (xtreemfs in this example) are correct.

Appendix B

Command Line Utilities

xtfs_mount The XtreemFS client which mounts an XtreemFS volume locally on a machine.

xtfs_umount Un-mounts a mounted XtreemFS volume.

xtfs_showmount Shows all locally mounted XtreemFS volumes.

xtfs_mkvol Creates a new volume on an MRC.

xtfs_lsvol Lists the volumes on an MRC.

xtfs_rmvol Deletes a volume and all files on that volume from the MRC and the OSDs.

xtfs_stat Displays XtreemFS specific file information such as the striping policy and the OSDs.

xtfs_sp Displays and modifies the striping policy for a file, or the default striping policy for directories and volumes.

xtfs_scrub Examines all files in a volume for incorrect file sizes and checksums. In case of incorrect file sizes, file sizes are corrected at the MRC.

xtfs_cleanup Deletes orphaned objects on an OSD or creates new metadata objects for orphaned files.

xtfs_mrcdbtool Dumps an XML representation of the MRC database to a given directory in the MRC's local file system.

Index