Project no. IST-033576

XtreemOS

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

Final Release of XtreemFS and OSS
D3.4.7

Due date of deliverable: 31-MAR-2010
Actual submission date: 09-APR-2010

Start date of project: June 1st 2006

Type: Deliverable
WP number: WP3.4

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

Version 1.0 / Last edited by Björn Kolbeck / 25-FEB-2010

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	25.02.10	Björn Kolbeck, Jan Stender	ZIB	Initial draft with updated XtreemFS User Guide
0.2	05.03.10	Florian Müller	UDUS	updated OSS User Guide
0.3	18.03.10	Björn Kolbeck	ZIB	List changes/new features
0.4	24.03.10	Kim-Thomas Möller	UDUS	Added descriptions of changes/new features in OSS
0.5	09.04.10	Jan Stender	ZIB	Made final changes according to Nicolas' and Sandrine's comments

Reviewers:
Nicolas Vigier (EDGE-IT), Zhouyi Zhou (ICT)

Tasks related to this deliverable:

Task No.	Task description	Partners involved$^\circ$
T3.4.5	Object Sharing Service	UDUS*
T3.4.7	XtreemFS Client-Side Caching	NEC*
T3.4.8	XtreemFS File Replication	ZIB*
T3.4.9	XtreemFS Automatic Replica Management	BSC*
T3.4.10	XtreemFS Testing, Performance, Compatibility and Maintenance	CNR*
T3.4.11	XtreemFS Consistent Snapshots	ZIB*

$^\circ$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 document contains an updated version of the two user guides for the XtreemFS File System and for the Object Sharing Service (OSS).

The XtreemFS user guide describes the release 1.2.1 of XtreemFS, the final release within the XtreemOS project which is included in XtreemOS 2.1. The documentation provided in D3.4.6 has been revised. In particular, we have added a new HDFS-compatible XtreemFS client that simplifies using XtreemFS for cloud computing applications that are built with the Hadoop Framework (see Section A.3.1). We have also added Vivaldi network coordinates which are used to automatically select the closest replica for a client based on network latency (see Section 2.6.4). A list of changes and new features in XtreemFS 1.2.1 compared to previous versions can be found in Section 2.1.

The OSS user guide describes the release 0.5 of the Object Sharing Service. The user guide contains a description of the internal library interface as well as the installation and usage of OSS.

In comparison to OSS release 0.4, which has been described in D3.4.6, OSS 0.5 implements ultra-peer based transaction validation, multithreading support for transactions and a highly concurrent network layer. These new features are described in Section 3.2. Section 3.4.1 documents the changes to the internal library interface, and Section 3.5 contains some new performance measurements comparing different transaction commit protocols. The configuration options for OSS 0.5 can be found in Appendix B.2.

Chapter 2

The XtreemFS User Guide

2.1 Changes

This is a summary of the most important changes in release 1.2:

- **asynchronous MRC backups**
  This feature allows administrators to create a backup of the MRC database without interrupting operations. The MRC creates an instantanious snapshot of it's database and writes this snapshot to disk in a background process.

- **vivaldi**
  XtreemFS now includes modules for calculating Vivaldi network coordinates to reflect the latency between OSDs and clients. An OSD and replica selection policy for Vivaldi is also available. For details, see Sec. 2.6.4.

- **renamed binaries**
  We renamed most binaries to conform with Linux naming conventions, e.g. `xtfs_mount` is now `mount.xtreemfs`. However, we added links with the old names for compatibility. For a full list see Sec. A.4.

- **"Grid SSL" mode**
  In this mode, SSL is only used for authentication (handshake) and regular TCP is used for communication afterwards. For more details see Sec. 2.4.2.

- **the `xctl` utility**
  The new release includes a command line utility `xctl` for starting and

stopping the services. This tool is useful if you don't want a package based installation or if you don't have root privileges. This utility also runs the automatic XtreemFS tests.

- **experimental Hadoop file system driver**
  XtreemFS includes an experimental driver for Hadoop. With this driver XtreemFS can be used as a replacement for HDFS, it offers the same semantics and API. The HDFS driver is a native Java client and circumvents the VFS and FUSE layer.

- **native Java client**
  The Java client offers an API similar to Java's `java.io.File` and `java.io.RandomAccessFile`. In addition, the Java client offers methods to control advanced, XtreemFS specific features like striping and replication.

Summary of important changes in release 1.2.1:

- **server status**
  Each server (especially OSDs) have a persistent status which can be online or dead/removed. This status must be changed manually and is used by the scrubber tool to identify dead OSDs which have been removed from the system.

- **enhanced scrubber**
  The scrubber is now able to remove replicas which are stored on OSDs that are marked as dead/removed. The scrubber will create new replicas for that file if a complete replica still exists and a sufficient number of OSDs is available. In addition, the scrubber marks replicas as "complete" if they contain all objects of the original file.

2.2 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 (http://www.xtreemfs.org)

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

(c) install with `rpm -Uhv xtreemfs*-1.2.x.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. In openSUSE users must be in the group `trusted`, in Ubuntu in the group `fuse`. You may need to log out and log in 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:30638.

9. Create a new volume with the default settings:
   `mkfs.xtreemfs localhost/myVolume`

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

11. Mount XtreemFS on your computer:

    ```
    mount.xtreemfs localhost/myVolume ~/xtreemfs
    ```

12. Have fun ;-)

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

You can also mount this volume on remote 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! This hostname has to be used instead of `localhost` when mounting.

2.3 About XtreemFS

Since you decided to take a look at this user guide, you probably read or heard about XtreemFS and want to find out more. This chapter contains basic information about the characteristics and the architecture of XtreemFS.

2.3.1 What is XtreemFS?

XtreemFS is a file system for a variety of different use cases and purposes. Since it is impossible to categorize or explain XtreemFS in a single sentence, we introduce XtreemFS by means of its two most significant properties: *XtreemFS is a globally distributed and replicated file system.*

What makes XtreemFS a distributed file system? We consider a file system as *distributed* if files are stored across a number of servers rather than a single server or local machine. Unlike local or network file systems, a distributed file system aggregates the capacity of multiple servers. As a *globally distributed* file system, XtreemFS servers may be dispersed all over the world. The capacity can be increased and decreased by adding and removing servers, but from a user's perspective, the file system appears to reside on a single machine.

What makes XtreemFS a replicated file system? We call it a *replicated* file system because replication is one of its most prominent features. XtreemFS is capable of maintaining replicas of files on different servers. Thus, files remain accessible even if single servers, hard disks or network connections fail. Besides, replication yields benefits in terms of data rates and access times. Different replicas of a file can be accessed simultaneously on different servers, which may lead to a better performance compared to simultaneous accesses on a single server. By placing file replicas close the consuming users and applications in a globally distributed installation, the effects of network latency and bandwidth reduction in wide area networks can be mitigated. However, replication is transparent to users and applications that work with XtreemFS; the file system is capable of controlling the life cycle and access of replicas without the need for human intervention or modifications of existing applications.

2.3.2 Is XtreemFS suitable for me?

If you consider using XtreemFS, you may be a system administrator in search of a better and more flexible alternative to your current data management solution. Or you may be a private user in need of a file system that can be easily set up and accessed from any machine in the world. You might also be someone looking for an open-source solution to manage large amounts of data distributed across multiple sites. In any case, you will wonder if XtreemFS fulfills your requirements. As a basis for your decision, the following two paragraphs point out the characteristics of XtreemFS.

XtreemFS is ...

... an open source file system. It is distributed freely and can be used by anyone without limitations.

... a POSIX file system. Users can mount and access XtreemFS like any other common file system. Application can access XtreemFS via the standard file system interface, i.e. without having to be rebuilt against a specialized API. XtreemFS supports a POSIX-compliant access control model.

... a multi-platform file system. Server and client modules can be installed and run on different platforms, including most Linux distributions, Solaris, Mac OS X and Windows.

... a globally distributed file system. Unlike cluster file systems, an XtreemFS installation is not restricted to a single administrative domain or cluster. It can span the globe and may comprise servers in different administrative domains.

... a failure-tolerant file system. As stated in the previous section, replication can keep the system alive and the data safe. In this respect, XtreemFS differs from most other open-source file systems.

... a secure file system. To ensure security in an untrusted, worldwide network, all network traffic can be encrypted with SSL connections, and users can be authenticated with X.509 certificates.

... a customizable file system. Since XtreemFS can be used in different environments, we consider it necessary to give administrators the possibility of adapting XtreemFS to the specific needs of their users. Customizable policies make it possible change the behavior of XtreemFS in terms of authentication, access control, striping, replica placement, replica selection and others. Such policies can be selected from a set of predefined policies, or implemented by administrators and plugged in the system.

XtreemFS is not ...

... a high-performance cluster file system. Even though XtreemFS reaches acceptable throughput rates on a local cluster, it cannot compete with specialized cluster file systems in terms of raw performance numbers. Most such file systems have an optimized network stack and protocols, and a substantially larger development team. If you have huge amounts of data on a local cluster with little requirements but high throughput rates to them, a cluster file system is probably the better alternative.

... a replacement for a local file system. Even though XtreemFS can be set up and mounted on a single machine, the additional software stack degrades the performance, which makes XtreemFS a bad alternative.

2.3.3 Core Features

The core functionality of XtreemFS is characterized by a small set of features, which are explained in the following.

Distribution. An XtreemFS installation comprises multiple servers that may run on different nodes connected on a local cluster or via the Internet. Provided that the servers are reachable, a client module installed on any machine in the world can access the installation. A binary communication protocol based on Sun's ONC-RPC ensures an efficient communication with little overhead between clients and servers. XtreemFS ensures that the file system remains in a consistent state even if multiple clients access a common set of files and directories. Similar to NFS, it offers a close-to-open consistency model in the event of concurrent file accesses.

Replication. Since version 1.0, XtreemFS supports *read-only replication.* A file may have multiple replicas, provided that the it was explicitly made read-only before, which means that its content cannot be changed anymore. This kind of replication can be used to make write-once files available to many consumers, or to protect them from losses due to hardware failures.

Besides complete replicas that are immediately synchronized after having been created, XtreemFS also supports partial replicas that are only filled with content on demand. They can e.g. be used to make large files accessible to many clients, of which only parts need to be accessed.

Currently, XtreemFS does not support replication of mutable files. From a technical perspective, this is more challenging than read-only replication, since XtreemFS has to ensure that all replicas of a file remain consistent despite attempts to concurrently write different replicas, as well as network and component failures. However, we are planning on supporting full read-write replication with future XtreemFS releases.

Striping. To ensure acceptable I/O throughput rates when accessing large files, XtreemFS supports *striping*. A striped file is split into multiple chunks ("*stripes*"), which are stored on different storage servers. Since different stripes can be accessed in parallel, the whole file can be read or written with the aggregated network and storage bandwidth of multiple servers. XtreemFS currently supports the `RAID0` striping pattern, which splits a file up in a set of stripes of a fixed size, and distributes them across a set of storage servers in a round-robin fashion. The size of an individual stripe as well as the number of storage servers used can be configured on a per-file or per-directory basis.

Security. To enforce security, XtreemFS offers mechanisms for user authentication and authorization, as well as the possibility to encrypt network traffic.

Authentication describes the process of verifying a user's or client's identity. By default, authentication in XtreemFS is based on local user names and depends on the trustworthiness of clients and networks. In case a more secure solution is needed, X.509 certificates can be used.

Authorization describes the process of checking user permissions to execute an operation. XtreemFS supports the standard UNIX permission model, which allows for assigning individual access rights to file owners, owning groups and other users.

Authentication and authorization are policy-based, which means that different models and mechanisms can be used to authenticate and authorize users. Besides, the policies are pluggable, i.e. they can be freely defined and easily extended.

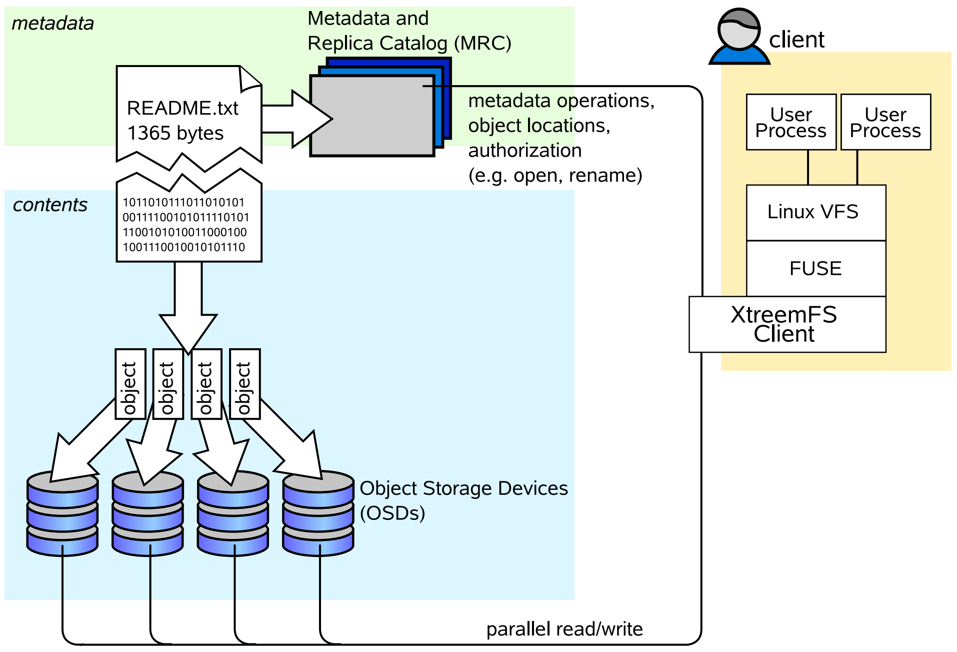


Figure 2.1: The XtreemFS architecture and components.

XtreemFS uses unauthenticated and unencrypted TCP connections by default. To encrypt all network traffic, services and clients can establish *SSL* connections. However, using SSL requires that all users and services have valid X.509 certificates.

2.3.4 Architecture

XtreemFS implements an *object-based file system architecture* (Fig. 2.1): file content is split into a series of fixed-size *objects* and stored across storage servers, while *metadata* is stored on a separate metadata server. The 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.

In contrast to block-based file systems, the management of available and used storage space is offloaded from the metadata server to the storage servers. Rather than inode lists with block addresses, file metadata contains lists of storage servers responsible for the objects, together with striping policies that define how to translate between byte offsets and object IDs. This implies that object sizes may vary from file to file.

XtreemFS Components. 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.
- 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.

2.4 XtreemFS Services

This chapter describes how to install and set up the server side of an XtreemFS installation.

2.4.1 Installation

When installing XtreemFS server components, 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*.

Note that the source tarball contains the complete distribution of XtreemFS, which also includes client and tools. Currently, binary distributions of the server are only available for Linux.

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-1.2.x.rpm xtreemfs-backend-1.2.x.rpm
```

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

```
$> dpkg -i xtreemfs-server-1.2.x.deb xtreemfs-backend-1.2.x.deb
```

To install the server components, the following package is required: `jre` $\geq$ 1.6.0 for RPM-based releases, `java6-runtime` for Debian-based releases. If you already have a different distribution of `Java6` on your system, you can alternatively install the XtreemFS server packages as follows:

```
$> rpm -i --nodeps xtreemfs-server-1.2.x.rpm \
   xtreemfs-backend-1.2.x.rpm
```

on RPM-based distributions,

```
$> dpkg -i --ignore-depends java6-runtime \
   xtreemfs-server-1.2.x.deb xtreemfs-backend-1.2.x.deb
```

on Debian-based distributions.

To ensure that your local `Java6` installation is used, is necessary to set the `JAVA_HOME` environment variable to your `Java6` installation directory, e.g.

```
$> export JAVA_HOME=/usr/java6
```

Both RPM and Debian-based packages will install three `init.d` scripts to start up the services (`xtreemfs-dir`, `xtreemfs-mrc`, `xtreemfs-osd`). If you want the services to be started automatically when booting up the system, you can execute `insserv <init.d script>` (SuSE), `chkconfig --add <init.d script>` (Mandriva, RedHat) or `update-rc.d <init.d script> defaults` (Ubuntu, Debian).

Installing from Sources

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

```
$> make server
```

This will build the XtreemFS server and Java-based tools. When done, execute

```
$> sudo make install-server
```

to install the server components. Finally, you will be asked to execute a post-installation script

```
$> sudo /etc/xos/xtreemfs/postinstall_setup.sh
```

to complete the installation.

2.4.2 Configuration

After having installed the XtreemFS server components, it is recommendable to configure the different services. This section describes the different configuration options.

XtreemFS services are configured via Java properties files that can be modified with a normal text editor. Default configuration files for a Directory Service, MRC and OSD are located in `/etc/xos/xtreemfs/`.

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 an MRC or OSD after it has been used for the first time!

The Directory Service resolves UUIDs to service endpoints, where each service endpoint consists of an IP address or hostname and port number. Each endpoint is associated with a netmask that indicates the subnet in which the mapping is valid. In theory, multiple endpoints can be assigned to a single

UUID if endpoints are associated with different netmasks. However, it is currently only possible to assign a single endpoint to each UUID; the netmask must be “*”, which means that the mapping is valid in all networks. Upon first start-up, OSDs and MRCs will auto-generate the mapping if it does not exist, by using 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 (time-to-live) of a mapping 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.

Automatic DIR Discovery

OSDs and MRCs are capable of automatically discovering a Directory Service. If automatic DIR discovery is switched on, the service will broadcast requests to the local LAN and wait up to 10s for a response from a DIR. The services will select the first DIR which responded, which can lead to non-deterministic behavior if multiple DIR services are present. Note that the feature works only in a local LAN environment, as broadcast messages are not routed to other networks. Local firewalls on the computers on which the services are running can also prevent the automatic discovery from working.

Security: The automatic discovery is a potential security risk when used in untrusted environments as any user can start-up DIR services.

A statically configured DIR address and port can be used to disable DIR discovery in the OSD and MRC (see Sec. 2.4.2, `dir_service`). By default. the DIR responds to UDP broadcasts. To disable this feature, set `discover = false` in the DIR service config file.

Authentication

Administrators may choose the way of authenticating users in XtreemFS. *Authentication Providers* are pluggable modules that determine how users are authenticated. For further details, see Sec. 2.7.1.

To set the authentication provider, it is necessary to set the following property in the MRC configuration file:

```
authentication_provider = <classname>
```

By default, the following class names can be used:

- `org.xtreemfs.common.auth.NullAuthProvider`
  uses local user and group IDs
- `org.xtreemfs.common.auth.SimpleX509AuthProvider`
  uses X.509 certificates; user and group IDs are extracted from the distinguished names of the certificates
- `org.xtreemos.XtreemOSAuthProvider`
  uses XOSCerts

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.

Note that it is not possible to mix SSL-enabled and non-SSL services in an XtreemFS installation! If you only need authentication based on certiciates without SSL, you can use the “grid SSL” mode. In this mode XtreemFS will only do an SSL handshake and fall back to plain TCP for communication. This mode is insecure (not encrypted and records are not signed) but just as fast as the non-SSL mode. If this mode is enabled, all client tools must be used with the `oncrpcg://` scheme prefix.

Each XtreemFS service needs a certificate and a private key in order to be run. Once they have been 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.

```
$> openssl 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. 2.4.2 for details).

```
ssl.enabled = 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. 2.4.3).

6. Create a new volume (see Sec. 2.5.2 for details).

```
$> mkfs.xtreemfs --pkcs12-file-path=\
   /etc/xos/xtreemfs/truststore/certs/client.p12 localhost/test
```

7. Mount the volume (see Sec. 2.5.3 for details).

```
$> mount.xtreemfs --pkcs12-file-path=\
   /etc/xos/xtreemfs/truststore/certs/client.p12 localhost/test /mnt
```

List of Configuration Options

All configuration parameters that may be used to define the behavior of the different services are listed in this section. Unless marked as optional, a parameter has to occur (exactly once) in a configuration file. Parameters marked as experimental belong to the DIR and MRC replication feature, which is currently under development. It is not recommended to mess about with these options if you want to use XtreemFS in production.

`admin_password` ***optional***

Services	DIR, MRC, OSD
Values	String
Default	
Description	Defines the admin password that must be sent to authorize requests like volume creation, deletion or shutdown. The same password is also used to access the HTTP status page of the service (user name is `admin`).

`authentication_provider`

Services	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. 2.4.2 for details.

`babudb.baseDir`

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.

`babudb.cfgFile` ***optional***

Services	DIR, MRC
Values	a file name
Default	DIR: `config.db` MRC: `config.db`
Description	Name for the database configuration file.

`babudb.checkInterval` ***optional***

Services	DIR, MRC
Values	a positive integer value
Default	DIR: `300` MRC: `300`
Description	The number of seconds between two checks of the disk log size for automatic checkpointing. Set this value to 0 to disable automatic checkpointing.

`babudb.compression` ***optional***

Services	DIR, MRC
Values	`true` or `false`
Default	DIR: `false` MRC: `false`
Description	Flag that determines whether the indices shall be compressed or not.

`babudb.debug.level` ***optional***

Services	DIR, MRC
Values	0, 1, 2, 3, 4, 5, 6, 7
Default	DIR: `4` MRC: `4`
Description	This is the debug level for BabuDB only. The debug level determines the amount and detail of information written to logfiles. Any debug level includes log messages from lower debug levels. The following log levels exist: 0 - fatal errors 1 - alert messages 2 - critical errors 3 - normal errors 4 - warnings 5 - notices 6 - info messages 7 - debug messages

`babudb.localTimeRenew` ***experimental, optional***

Services	DIR, MRC
Values	a positive integer value
Default	
Description	Intervall in milliseconds for synchronizing the ONCRPCServer and ONCRPCClient of the BabuDB replication with the local clock.

`babudb.logDir`

Services	DIR, MRC
Values	absolute file system path
Default	DIR: `/var/lib/xtreemfs/dir/db-log` MRC: `/var/lib/xtreemfs/mrc/db-log`
Description	The directory the MRC uses to store database logs. 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.

`babudb.maxLogfileSize` ***optional***

Services	DIR, MRC
Values	a positive integer value
Default	DIR: `16777216` MRC: `16777216`
Description	If automatic checkpointing is enabled, a checkpoint is created when the disk logfile exceedes maxLogfileSize bytes. The value should be reasonable large to keep the checkpointing-rate low. However, it should not be too large as a large disk log increases the recovery time after a crash.

`babudb.pseudoSyncWait` ***optional***

Services	DIR, MRC
Values	a positive integer value
Default	DIR: `200` MRC: `0`
Description	The BabuDB disk logger can batch multiple operations into a single write+fsync to increase the throughput. This does only work if there are operations executed in parallel by the worker threads. In turn, if you work on a single database it becomes less efficient. To circumvent this problem, BabuDB offers a pseudo-sync mode which is similar to the PostgreSQL write-ahead log (WAL). If pseduoSyncWait is set to a value larger then 0, this pseudo-sync mode is enabled. In this mode, insert operations are acknowledged as soon as they have been executed on the in-memory database index. The disk logger will execute a batch write of up to 500 operations followed by a single sync (see syncMode) every pseudoSyncWait ms. This mode is considerably faster than synchronous writes but you can lose data in case of a crash. In contrast to ASYNC mode the data loss is limited to the operations executed in the last pseudoSyncWait ms.

`babudb.repl.backupDir` ***experimental, optional***

Services	DIR, MRC
Values	a absolute file system path
Default	
Description	DB backup directory needed for the initial loading of a slave BabuDB from the master BabuDB.

`babudb.repl.chunkSize` ***experimental, optional***

Services	DIR, MRC
Values	a positive integer greater 0
Default	
Description	At the initial load mechanism the database files will be split into chunks. The size of this chunks in bytes can be defined here.

`babudb.repl.participant` ***experimental, optional***

Services	DIR, MRC
Values	see description
Default	
Description	The addresses of the hosts running an instance of BabuDB and participating at the replication. The local address is also required in this list. BabuDB will find out on its own, which of the given addresses is the local one. Field babudb.repl.participant.i defines the domain name or IP address. Field babudb.repl.participant.i.port defines the port. babudb.repl.participant.0 babudb.repl.participant.0.port babudb.repl.participant.1 babudb.repl.participant.1.port ... babudb.repl.participant.n babudb.repl.participant.n.port for n participants.

`babudb.repl.sync.n` ***experimental, optional***

Services	DIR, MRC
Values	a positive integer value
Default	
Description	The number of slaves that at least have to be synchronous to the master BabuDB application. To use asynchronous replication, this value has to be set to 0. Otherwise it has to be less or equal to the number of participants.

`babudb.ssl.authenticationWithoutEncryption` ***experimental, optional***

Services	DIR, MRC
Values	`true` or `false`
Default	
Description	Flag that determines if authentication should be used without encrypting the connection. For BabuDB replication only.

`babudb.ssl.enabled` ***experimental, optional***

Services	DIR, MRC
Values	`true` or `false`
Default	
Description	Options to use a secure connection with SSL authenticaion and optionally encryption for the BabuDB replication.

`babudb.ssl.service_creds` ***experimental, optional***

Services	MRC, OSD
Values	path to file
Default	
Description	Must be specified if `babudb.ssl.enabled` is `true`. Specifies the file containing the service credentials (X.509 certificate and private key). PKCS#12 and JKS format can be used, set `babudb.ssl.service_creds.container` accordingly. This file is used during the SSL handshake to authenticate the service.

`babudb.ssl.service_creds.container` ***experimental, optional***

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

`babudb.ssl.service_creds.pw` ***experimental, optional***

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

`babudb.ssl.trusted_certs` ***experimental, optional***

Services	MRC, OSD
Values	path to file
Default	
Description	Must be specified if `babudb.ssl.enabled` is `true`. Specifies the file containing the trusted root certificates (e.g. CA certificates) used to authenticate clients.

`babudb.ssl.trusted_certs.container` ***experimental, optional***

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

`babudb.ssl.trusted_certs.pw` ***experimental, optional***

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

`babudb.sync`

Services	DIR, MRC
Values	ASYNC, SYNC_WRITE_METADATA, SYNC_WRITE, FDATASYNC or FSYNC
Default	DIR: `FSYNC` MRC: `ASYNC`
Description	The sync mode influences how operations are commited to the disk log before the operation is acknowledged to the caller.

- ASYNC mode the writes to the disk log are buffered in memory by the operating system. This is the fastest mode but will lead to data loss in case of a crash, kernel panic or power failure.

- SYNC_WRITE_METADATA opens the file with O_SYNC, the system will not buffer any writes. The operation will be acknowledged when data has been safely written to disk. This mode is slow but offers maximum data safety. However, BabuDB cannot influence the disk drive caches, this depends on the OS and hard disk model.

- SYNC_WRITE similar to SYNC_WRITE_METADATA but opens file with O_DSYNC which means that only the data is commit to disk. This can lead to some data loss depending on the implementation of the underlying file system. Linux does not implement this mode.

- FDATASYNC is similar to SYNC_WRITE but opens the file in asynchronous mode and calls fdatasync() after writing the data to disk.

- FSYNC is similar to SYNC_WRITE_METADATA but opens the file in asynchronous mode and calls fsync() after writing the data to disk.

For best throughput use ASYNC, for maximum data safety use FSYNC.

`babudb.worker.maxQueueLength` ***optional***

Services	DIR, MRC
Values	a positiv integer value
Default	DIR: `250` MRC: `250`
Description	If set to a value larger than 0, this is the maximum number of requests which can be in a worker's queue. This value should be used if you have pseudo-synchronous mode enabled to ensure that your queues don't grow until you get an out of memory exception. Can be set to 0 if pseudo-sync mode is disabled.

`babudb.worker.numThreads` ***optional***

Services	DIR, MRC
Values	a positiv integer value
Default	DIR: `0` MRC: `0`
Description	The number of worker threads to be used for database operations. As BabuDB does not use locking, each database is handled by only one worker thread. If there are more databases than worker threads, the databases are distributed onto the available threads. The number of threads should be set to a value smaller than the number of available cores to reduce overhead through context switches. You can also set the number of worker threads to 0. This will considerably reduce latency, but may also decrease throughput on a multicore system with more than one database.

`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.

`capability_timeout` ***optional***

Services	MRC
Values	seconds
Default	600
Description	Defines the relative time span for which a capability is valid after having been issued.

`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
Default	Adler32
Description	Must be specified if `checksums.enabled` is enabled. This property defines the algorithm used to create OSD checksums.

`debug.level` ***optional***

Services	DIR, MRC, OSD
Values	0, 1, 2, 3, 4, 5, 6, 7
Default	6
Description	The debug level determines the amount and detail of information written to logfiles. Any debug level includes log messages from lower debug levels. The following log levels exist:

0 - fatal errors

1 - alert messages

2 - critical errors

3 - normal errors

4 - warnings

5 - notices

6 - info messages

7 - debug messages

`debug.categories` ***optional***

Services	DIR, MRC, OSD
Values	all, lifecycle, net, auth, stage, proc, db, misc
Default	all
Description	Debug categories determine the domains for which log messages will be printed. By default, there are no domain restrictions, i.e. log messages form all domains will be included in the log. The following categories can be selected:

all -	no restrictions on the category
lifecycle -	service lifecycle-related messages, including startup and shutdown events
net -	messages pertaining to network traffic and communication between services
auth -	authentication and authorization-related messages
stage -	messages pertaining to the flow of requests through the different stages of a service
proc -	messages about the processing of requests
db -	messages that are logged in connection with database accesses
misc -	any other log messages that do not fit in one of the previous categories

Note that it is possible to specify multiple categories by means of a comma or space-separated list.

`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. If set to `.autodiscover` the service will use the automatic DIR discovery mechanism (see Sec. 2.4.2). (Note that the initial '.' is used to avoid ambiguities with hosts called "autodiscover".)

`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.

`discover` ***optional***

Services	DIR
Values	true, false
Default	true
Description	If set to true the DIR will received UDP broadcasts and advertise itself in response to XtreemFS components using the DIR automatic discovery mechanism. If set to false, the DIR will ignore all UDP traffic. For details see Sec. 2.4.2.

`geographic_coordinates` ***optional***

Services	DIR, MRC, OSD
Values	String
Default	
Description	Specifies the geographic coordinates which are registered with the directory service. Used e.g. by the web console.

`hostname` ***optional***

Services	MRC, OSD
Values	String
Default	
Description	If specified, it defines the host name that is used to register the service at the directory service. If not specified, the host address defined in `listen.address` will be used if specified. If neither `hostname` nor `listen.address` are specified, the service itself will search for externally reachable network interfaces and advertise their addresses.

`http_port`

Services	DIR, MRC, OSD
Values	1 .. 65535
Default	30636 (MRC), 30638 (DIR), 30640 (OSD)
Description	Specifies the listen port for the HTTP service that returns the status page.

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

Services	OSD
Values	IP address
Default	
Description	If specified, it 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 ONC-RPC connections (TCP). The OSD uses the specified port for both TCP and UDP. Please make sure to configure your firewall to allow incoming TCP traffic (plus UDP traffic, in case of an OSD) 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.

`monitoring`

Services	DIR
Values	true, false
Default	false
Description	Enables the built-in monitoring tool in the directory service. If enabled, the DIR will send alerts via emails if services are crashed (i.e. do not send heartbeat messages). No alerts will be sent for services which signed-off at the DIR. To enable monitoring you also need to configure `email.receiver`, `email.program`. In addition, you may want to change the values for `email.sender`, `email.programm`, `service_timeout_s` and `max_warnings`.

`monitoring.email.programm`

Services	DIR
Values	path
Default	`/usr/sbin/sendmail`
Description	Location of the `sendmail` binary to be used for sending alert mails. See `monitoring`.

`monitoring.email.receiver`

Services	DIR
Values	email address
Default	-
Description	Email address of recipient of alert emails. See `monitoring`.

`monitoring.email.sender`

Services	DIR
Values	email address
Default	"XtreemFS DIR service ¡dir@localhost¿"
Description	Email address and sender name to use for sending alert mails. See `monitoring`.

`monitoring.max_warnings`

Services	DIR
Values	0..N
Default	1
Description	Number of alert mails to send for a single service which has crashed/disconnected. Each alert mail contains a summary of all crashed/disconnected services. See `monitoring`.

`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.

`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. 2.7.3). 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. 2.7.3).

`service_timeout_s`

Services	DIR
Values	0..N seconds
Default	300
Description	Time to wait for a heartbeat message before sending an alert email. See `monitoring`.

`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.grid_ssl`

Services	DIR, MRC, OSD
Values	true, false
Default	false
Description	In this mode the services and client will only use SSL for mutual authentication with X.509 certificates (SSL handshake). After successful authentication the communication is via plain TCP. This means that there is no encryption and signing of records! This mode is comparable to HTTP connections with Digest authentication. It should be used when certificate based authentication is required but performance is more important than security, which is usually true in GRID installations. If this mode is enabled, all client tools must be used with the `oncrpcg://` scheme prefix.

`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`.

`startup.wait_for_dir`

Services	MRC, OSD
Values	0..N seconds
Default	30
Description	Time to wait for the DIR to become available during start up of the MRC and OSD. If the DIR does not respond within this time the MRC or OSD will abort startup.

`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`.

2.4.3 Execution and Monitoring

This section describes how to execute and monitor XtreemFS services.

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
```

To run init.d scripts, root permissions are required. **Note** that the Directory Service must be started first, since a running Directory Service is required when starting an MRC or OSD. Once a Directory Service as well as at least one OSD and MRC are running, XtreemFS is operational.

Alternatively, you can use the `xctl` tool to start and stop the services. Config files are read from `./etc/xos/xtreemfs` unless another path is specified with `-c <path>`.

```
$> bin/xctl --start-dir
$> bin/xctl --start-mrc
$> bin/xctl --start-osd
```

or

```
$> bin/xctl --stop-osd
$> bin/xctl --stop-mrc
$> bin/xctl --stop-dir
```

The servers will be executed under the user ID of the user who called the xctl script.

Web-based Status Page

Each XtreemFS service can generate an HTML status page, which displays runtime information about the service (Fig. 2.2). The HTTP server that generates the status page runs on the port defined by the configuration property `http_port`; default values are 30636 for MRCs, 30638 for Directory Services, and 30640 for OSDs.

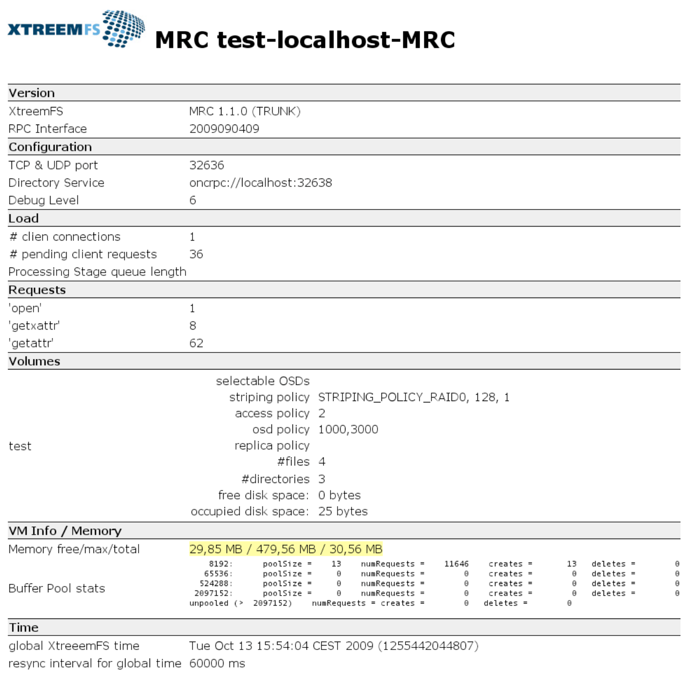
The status page of an MRC can e.g. be shown by opening

`http://my-mrc-host.com:30636/`

with a common web browser. If you set an admin password in the service's configuration, you will be asked for authentication when accessing the status page. Use `admin` as username.

DIR Service Monitoring

The directory service has a built-in notification system that can send alert emails if a service fails to send heartbeat messages for some time. The monitoring can be enabled in the DIR configuration by setting `monitoring = true`.



MRC test-localhost-MRC

Version	
XtreemFS	MRC 1.1.0 (TRUNK)
RPC Interface	2009090409
Configuration	
TCP & UDP port	32636
Directory Service	oncrpc://localhost:32638
Debug Level	6
Load	
# clien connections	1
# pending client requests	36
Processing Stage queue length	
Requests	
'open'	1
'getxattr'	8
'getattr'	62
Volumes	
test	selectable OSDs striping policy STRIPING_POLICY_RAID0, 128, 1 access policy 2 osd policy 1000,3000 replica policy #files 4 #directories 3 free disk space: 0 bytes occupied disk space: 25 bytes
VM Info / Memory	
Memory free/max/total	29,85 MB / 479,56 MB / 30,56 MB
Buffer Pool stats	see below
Time	
global XtreeemFS time	Tue Oct 13 15:54:04 CEST 2009 (1255442044807)
resync interval for global time	60000 ms

```
    8192:      poolSize =    13    numRequests =    11646    creates =       13   deletes =        0
   65536:      poolSize =     0    numRequests =        0    creates =        0   deletes =        0
  524288:      poolSize =     0    numRequests =        0    creates =        0   deletes =        0
 2097152:      poolSize =     0    numRequests =        0    creates =        0   deletes =        0
unpooled (>  2097152)    numRequests = creates =        0   deletes =        0
```

Figure 2.2: OSD status web page

2.4.4 Troubleshooting

Various issues may occur when attempting to set up an XtreemFS server component. If a service fails to start, the log file often reveals useful information. Server log files are located in `/var/log/xtreemfs`. Note that you can restrict granularity and categories of log messages via the configuration properties `debug.level` and `debug.categories` (see Sec. 2.4.2).

If an error occurs, please check if one of the following requirements is not met:

- You have root permissions when starting the service. Running the `init.d` scripts requires root permissions. However, the services themselves are started on behalf of a user *xtreemfs*.
- DIR has been started before MRC and OSD. Problems may occur if a script starts multiple services as background processes.
- There are no firewall restrictions that keep XtreemFS services from communicating with each other. The default ports that need to be open are: 32636 (MRC, TCP), 32638 (DIR, TCP), and 32640 (OSD, TCP & UDP).

- The MRC database version is correct. In case of an outdated database version, the `xtfs_mrcdbtool` commands of the old and new XtreemFS version can dump and restore the database, respectively (see Sec. 2.6.2).
- A network interface is available on the host. It may be either bound to an IPv4 or IPv6 address.

2.5 XtreemFS Client

The XtreemFS client is needed to access an XtreemFS installation from a remote machine. This chapter describes how to use the XtreemFS client in order to work with XtreemFS like a local file system.

2.5.1 Installation

There are two different installation sources for the XtreemFS Client: *pre-packaged releases* and *source tarballs*.

Note that the source tarball contains the complete distribution of XtreemFS, which also includes server and tools. Currently, binary distributions of the client are only available for Linux and Windows.

Prerequisites

To install XtreemFS on Linux, please make sure that FUSE 2.6 or newer, openSSL 0.9.8 or newer and a Linux 2.6 kernel are available on your system. For an optimal performance, we suggest to use FUSE 2.8 with a kernel version 2.6.26 or newer.

To build the Linux XtreemFS Client from the source distribution, you also need the openSSL headers (e.g. openssl-devel package), python $\geq$ 2.4, and gcc-c++ $\geq$ 4.2.

Installing from Pre-Packaged Releases

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

```
$> rpm -i xtreemfs-client-1.2.x.rpm
```

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

```
$> dpkg -i xtreemfs-client-1.2.x.deb
```

For Windows, please use the *.msi* installer that will guide you through the installation process.

Installing from Sources

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

```
$> make client
```

This will build the XtreemFS client and non-Java-based tools. Note that the following third-party packages are required on Linux:

```
python >= 2.4
gcc-c++ >= 4
fuse >= 2.6
fuse-devel >= 2.6 (RPM-based distros)
libfuse-dev >= 2.6 (DEB-based distros)
libopenssl-devel >= 0.8 (RPM-based distros)
libssl-dev >= 0.9 (DEB-based distros)
```

When done, execute

```
$> sudo make install-client
```

to complete the installation of XtreemFS.

2.5.2 Volume Management

Like many other file systems, XtreemFS supports the concept of volumes. A volume can be seen as a container for files and directories with its own policy settings, e.g. for access control and replication. Before being able to access an XtreemFS installation, at least one volume needs to be set up. This section describes how to deal with volumes in XtreemFS.

Creating Volumes

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

When creating a volume, it is recommended to specify the access control policy (see Sec. 2.7.2). If not specified, POSIX permissions/ACLs will be chosen by default. Unlike most other policies, access control policies cannot be changed afterwards.

In addition, it is recommended to set a default striping policy (see Sec. 2.7.4). If no per-file or per-directory default striping policy overrides the volume's default striping policy, the volume's policy is assigned to all newly created files. If no volume policy is explicitly defined when creating a volume, a RAID0 policy with a stripe size of 128kB and a width of 1 will be used as the default policy.

A volume with a POSIX permission model, a stripe size of 256kB and a stripe width of 1 (i.e. all stripes will reside on the same OSD) can be created as follows:

```
$> mkfs.xtreemfs -a POSIX -p RAID0 -s 256 -w 1 \
   my-mrc-host.com:32636/myVolume
```

Creating a volume may require privileged access, which depends on whether an administrator password is required by the MRC. To pass an administrator password, add `--password <password>` to the `mkfs.xtreemfs` command.

For a complete list of parameters, please refer to the `mkfs.xtreemfs` man page.

Deleting Volumes

Volumes can be deleted with the `rmfs.xtreemfs` tool. Note that deleting a volume implies that *any data, i.e. all files and directories on the volume are deleted*! Please see `man rmfs.xtreemfs` for a full list of options and usage.

The volume `myVolume` residing on the MRC `my-mrc-host.com:32636` can e.g. be deleted as follows:

```
$> rmfs.xtreemfs my-mrc-host.com:32636/myVolume
```

Volume deletion is restricted to volume owners and privileged users. Similar to `xtfs_mkvol`, an administrator password can be specified if required.

Listing all Volumes

A list of all volumes can be displayed with the `lsfs.xtreemfs` tool. All volumes hosted by the MRC `my-mrc-host.com:32636` can be listed as follows:

```
$> lsfs.xtreemfs my-mrc-host.com:32636
```

Adding the `--l` flag will result in more details being shown.

2.5.3 Accessing Volumes

Once a volume has been created, it needs to be mounted in order to be accessed.

Mounting and Un-mounting

Before mounting XtreemFS volumes on a Linux machine, 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 file systems.

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

Volumes are mounted with the `xtfs_mount` command:

```
$> mount.xtreemfs remote.dir.machine/myVolume /xtreemfs
```

`remote.dir.machine` describes the host with the Directory Service at which the volume is registered; `myVolume` is the name of the volume to be mounted. `/xtreemfs` is the directory on the local file system to which the XtreemFS volume will be mounted. For more options, please refer to `man mount.xtreemfs`.

Please be aware that the Directory Service URL needs to be provided when mounting a volume, while MRC URLs are used to create volumes.

When mounting a volume, the client will immediately go into background and won't display any error messages. Use the `-f` option to prevent the mount process from going into background and get all error messages printed to the console.

To check that a volume is mounted, use the `mount` command. It outputs a list of all mounts in the system. XtreemFS volumes are listed as `type fuse`:

```
xtreemfs on /xtreemfs type fuse (rw,nosuid,nodev,user=userA)
```

Volumes are unmounted with the `umount.xtreemfs` tool:

```
$> umount.xtreemfs /xtreemfs
```

Mount Options

Access to a FUSE mount is usually restricted to the user who mounted the volume. To allow the root user or any other user on the system to access the mounted volume, the FUSE options `-o allow_root` and `-o allow_other` can be used with `xtfs_mount`. 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`.

By default, the local system cache on the client machine will be used to speed up read access to XtreemFS. In particular, using the cache as a local buffer is necessary to support the `mmap` system call, which - amongst others - is required to execute applications on Linux. On the other hand, using buffered I/O may adversely affect throughput when writing large files, as FUSE $\leq$ 2.7 splits up large writes into multiple individual 4k (page size) writes. In addition, it limits the consistency model of client caches to "close-to-open", which is similar to the model provided by NFS. Buffered I/O can be switched off by adding the `-o direct_io` parameter. The parameter effects that all read and write operations are directed to their OSDs instead of being served from local caches.

2.5.4 Troubleshooting

Different kinds of problems may occur when trying to create, mount or access files in a volume. If no log file was specified, the client will create a logfile called `mount.xtreemfs.log` in the current working direcory. This logile is only created in case of an error message. In case no useful error message is printed on the console or in the logfile, it may help to enable client-side log output. This can be done as follows:

```
$> mount.xtreemfs -f -d INFO remote.dir.machine/myVolume /xtreemfs
```

The following list contains the most common problems and their solutions.

Problem **A volume cannot be created or mounted.**

Solution Please check your firewall settings on the server side. Are all ports accessible? The default ports are 32636 (MRC), 32638 (DIR), and 32640 (OSD).
In case the XtreemFS installation has been set up behind a NAT, it is possible that services registered their NAT-internal network interfaces at the DIR. In this case, clients cannot properly resolve server addresses, even if port forwarding is enabled. Please check the *Address Mappings* section on the DIR status page to ensure that externally reachable network interfaces have been registered for the your servers' UUIDs. If this is not the case, it is possible to explicitly specify the network interfaces to register via the `hostname` property (see Sec. 2.4.2).

Problem **When trying to mount a volume,** `ONC-RPC exception: system error` **appears on the console.**

Solution The most common reason are incompatible protocol versions in client and server. Please make sure that client and server have the same release version numbers. They can be determined as follows:

Server: check the status pages. Alternatively, execute `rpm -q xtreemfs-server` on RPM-based distributions, or `dpkg -l | grep xtreemfs-server` on DEB-based distributions.

Client: execute `rpm -q xtreemfs-client` on RPM-based distributions, or `dpkg -l | grep xtreemfs-client` on DEB-based distributions.

Problem **An error occurs when trying to access a mounted volume.**

Solution Please make sure that you have sufficient access rights to the volume root. Superusers and volume owners can change these rights via `chmod <mode> <mountpoint>`. If you try to access a mount point to which XtreemFS was mounted by a different user, please make sure that the volume is mounted with `xtfs_mount -o allow_other ...`.

Problem **An I/O error occurs when trying to create new files.**

Solution In general, you can check the contents of the client log file to see the error which caused the I/O error. A common reason for this problem is that no OSD could be assigned to the new file. Please check if suitable OSDs are available for the volume. There are two alternative ways to do this:

- Open the MRC status page. It can be accessed via `http://<MRC-host>:30636` in the default case. For each volume, a list of suitable OSDs is shown there.
- Execute `getfattr -n xtreemfs.usable_osds --only-values <mountpoint>`.

There may be different reasons for missing suitable OSDs:

- One or more OSDs failed to start up. Please check the log files and status pages of all OSDs to ensure that they are running.
- One or more OSDs failed to register or regularly report activity at the DIR. Please check the DIR status page to ensure that all OSDs are registered and active.
- There are no OSDs with a sufficient amount of free disk space. Please check the OSD status page to obtain information about free disk space.

Problem	**An I/O error occurs when trying to access an existing file.**
Solution	Please check whether all OSDs assigned to the file are running and reachable. This can be done as follows: 1. Get the list of all OSDs for the file: `getfattr -n xtreemfs.locations --only-values <file>`. 2. Check whether the OSDs in (one of) all replicas in the list are running and reachable, e.g. by opening the status pages or via `telnet <host> <port>`.

2.6 XtreemFS Tools

To make use of most of the advanced XtreemFS features, XtreemFS offers a variety of different tools. There are tools that support administrators with the maintenance of an XtreemFS installation, as well as tools for controlling features like replication and striping. An overview of the different tools with descriptions of how to use them are provided in the following.

2.6.1 Installation

When installing the XtreemFS tool suite, 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*.

Note that the source tarball contains the complete distribution of XtreemFS, which also includes client and server. Currently, binary distributions of the tools are only available for Linux.

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-tools-1.2.x.rpm xtreemfs-backend-1.2.x.rpm
```

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

```
$> dpkg -i xtreemfs-tools-1.2.x.deb xtreemfs-backend-1.2.x.deb
```

To install the tools, the following package is required: `jre` $\geq$ 1.6.0 for RPM-based releases, `java6-runtime` for Debian-based releases. If you already have a different distribution of `Java6` on your system, you can alternatively install the XtreemFS tools packages as follows:

```
$> rpm -i --nodeps xtreemfs-tools-1.2.x.rpm \
   xtreemfs-backend-1.2.x.rpm
```

on RPM-based distributions,

```
$> dpkg -i --ignore-depends java6-runtime \
   xtreemfs-tools-1.2.x.deb xtreemfs-backend-1.2.x.deb
```

on Debian-based distributions.

To ensure that your local `Java6` installation is used, is necessary to set the `JAVA_HOME` environment variable to your `Java6` installation directory, e.g.

```
$> export JAVA_HOME=/usr/java6
```

All XtreemFS tools will be installed to `/usr/bin`.

Installing from Sources

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

```
$> make server
```

When done, execute

```
$> sudo make install-tools
```

to complete the installation. Note that this will also install the XtreemFS client and servers.

2.6.2 Maintenance Tools

This section describes the tools that support administrators in maintaining an XtreemFS installation.

MRC Database Conversion

The database format in which the MRC stores its file system metadata on disk may change with future XtreemFS versions, even though we attempt to keep it as stable as possible. To ensure that XtreemFS server components may be updated without having to create and restore a backup of the entire installation, it is possible to convert an MRC database to a newer version by means of a version-independent XML representation.

This is done as follows:

1. Create an XML representation of the old database with the old MRC version.
2. Update the MRC to the new version.
3. Restore the database from the XML representation.

`xtfs_mrcdbtool` is a tool that is capable of doing this. It can create an XML dump of an MRC database as follows:

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

A file `dump.xml` containing the entire database content of the MRC running on `my-mrc-host.com:32636` is written to `/tmp/dump.xml`. For security reasons, the dump file will be created locally on the MRC host. To make sure that sufficient write permissions are granted to create the dump file, we therefore recommend to specify an absolute dump file path like `/tmp/dump.xml`.

A database dump can be restored from a dump file as follows:

```
$> xtfs_mrcdbtool -mrc oncrpc://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.

Please be aware that dumping and restoring databases may both require privileged access rights if the MRC requires an administrator password. The password can be specified via `--p`; for further details, check the `xtfs_mrcdbtool` man page.

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 oncrpc://my-dir-host.com:32638 myVolume
```

This will scrub each file in the volume `myVolume`, i.e. check file size consistency and set the correct 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 scans an OSD for orphaned objects, which can be used as follows:

```
$> xtfs_cleanup -dir oncrpc://localhost:32638 \
   uuid:u2i3-28isu2-iwuv29-isjd83
```

The given UUID identifies the OSD to clean and will be resolved by the directory service defined by the `-dir` option (localhost:32638 in this example). The process will be started and can be stopped by setting the option `-stop`. To watch the cleanup progress use option `-i` for the interactive mode. For further information see `man xtfs_cleanup`.

Setting the Service Status

The service's status field is shown in the service status page as `static.status`. The status can be 0 (online), 1 (marked for removal) and 2 (dead/removed). Status 0 (online) is the regular status for all services, even if they are temporarily offline. Status 2 (dead/removed) marks an OSD as permanently failed and the scrubber will removed replicas and files from these OSDs. Status 1 (marked for removal) is for future use.

The status can be set with the `xtfs_chstatus` tool:

```
$> xtfs_chstatus -dir oncrpc://localhost:32638 \
   u2i3-28isu2-iwuv29-isjd83 online
```

This command sets the status of the service with the UUID `u2i3-28isu2-iwuv29-isjd83` to online.

2.6.3 User Tools

Besides administrator tools, a variety of tools exist that make advanced XtreemFS features accessible to users. These tools will be described in this section.

Showing 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              oncrpc://localhost/test/test.txt
XtreemFS fileID           41e9a04d-0b8b-467b-94ef-74ade02a2dc9:6
object type               regular file
owner                     stender
group                     users
read-only                 false

XtreemFS replica list
   list version           0
   replica update policy
   -----------------------------
   replica 1 SP           STRIPING_POLICY_RAID0, 128kb, 1
   replica 1 OSDs         [{address=127.0.0.1:32640, uuid=OSD1}]
   replica 1 repl. flags  0x1
   -----------------------------
```

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.

Changing Striping Policies

Currently, it is not possible to change the striping policy of an existing file, as this would require rearrangements and transfers of data between OSDs.

However, it is possible to define individual striping policies for files that will be created in the future. This can be done by changing the default striping policy of the parent directory or volume.

XtreemFS provides the `xtfs_sp` tool. The tool can be used to change the striping policy that will be assigned to newly created files as follows:

```
$> xtfs_sp --set -p RAID0 -w 4 -s 256 /xtreemfs/dir
```

This will cause a RAID0 striping policy with 256kB stripe size and four OSDs to be assigned to all newly created files in `/xtreemfs/dir`.

The tool can display the default striping policy of a volume or directory as follows:

```
$> xtfs_sp --get /xtreemfs/dir
```

This will result in output similar to the following:

```
file:         /xtreemfs/dir
policy:       STRIPING_POLICY_RAID0
stripe-size:  4
width (kB):   256
```

When creating a new file, XtreemFS will first check whether a default striping policy has been assigned to the file's 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 volume or directory.

Read-Only Replication

Replication is one of core features of XtreemFS. A replica can be seen as a (not essentially complete) copy of a file's content on a remote (set of) OSD(s). Replication is handled among the XtreemFS OSDs, which makes it completely transparent to client applications.

So far, XtreemFS only supports *read-only replication*. Read-only replication requires files to be immutable (i.e. 'read-only'), which implies that once a file has been replicated, it can no longer be modified. The benefit of read-only replicas is that XtreemFS can guarantee sequential replica consistency at a low cost; since files are no longer modified when replicated, no overhead is caused to ensure replica consistency.

When replicating a file, the first step is to make the file read-only, which can be done as follows:

```
$> xtfs_repl --set_readonly local-path-of-file
```

Once a file has been marked as read-only, replicas can be added. The tool supports different replica creation modes. The automatic mode retrieves a list of OSDs from the MRC and chooses the best OSD according to the current replica selection policy. You can also select a specific OSD by specifying its UUID on the command line.

Newly created replicas are initially empty, which means that no file content has been copied from other non-empty replicas. Yet, they can be immediately used by applications. If a replica does not have the requested data, it fetches the data from a remote replica and saves it locally for future requests (on-demand replication). Such partial replicas help to save network bandwidth and disk usage. Alternatively, replicas can be triggered to fetch the whole data from remote replicas in the background, regardless of client requests (background replication).

Moreover, XtreemFS supports different transfer strategies which has a big impact on the speed of the replication and the order in which objects are fetched. A transfer strategy must be chosen for each replica.

A replica can e.g. created as follows:

```
$> xtfs_repl --add_auto --full --strategy random \
   /xtreemfs/file.txt
```

This command creates a new replica with an automatically-selected set of OSDs (for details, see Sec. 2.7.3, 2.6.3). The switch `--full` indicates that background replication is desired; otherwise, replicas are filled on demand, which means that they remain partial replicas until the application accesses all the objects of the replica.

To list all replicas and OSDs of the file use:

```
$> xtfs_repl -l /xtreemfs/file.txt
```

This generates output similar to this:

```
File is read-only.
REPLICA 1:
         Striping Policy: STRIPING_POLICY_RAID0
         Stripe-Size: 128,00 kB
         Stripe-Width: 1 (OSDs)
         Replication Flags:
                 Complete: false
                 Replica Type: partial
                 Transfer-Strategy: random
         OSDs:
                 [Head-OSD]     UUID: osd1, URL: /127.0.0.1:32641
REPLICA 2:
         Striping Policy: STRIPING_POLICY_RAID0
         Stripe-Size: 128,00 kB
         Stripe-Width: 1 (OSDs)
         Replication Flags:
                 Complete: true
                 Replica Type: partial
                 Transfer-Strategy: unknown
         OSDs:
                 [Head-OSD]     UUID: osd2, URL: /127.0.0.1:32640
```

Besides adding replicas, replicas can also be removed. Since replicas of a file do not have a fixed order, we use a replica's first OSD to identify the replica to delete. The first OSD in a replica's list of OSDs, also referred to as *head OSD* is a unique identifier for a replica, as different replicas of a file may not share any OSDs.

To remove a replica, the UUID of the head OSD must be given as an argument. It can be determined via `xtfs_repl -l`. To ensure that at least one complete replica remains, i.e. a replica that stores the entire file content, complete replicas can only be removed if there is at least one more complete complete replica of the file.

A replica can be removed as follows:

```
$> xtfs_repl -r osd1 /xtreemfs/file.txt
```

`osd1` refers to the UUID of the head OSD in the replica to remove.

Automatic On-Close Replication

In addition to manually adding and removing replicas, XtreemFS supports an automatic creation of new replicas when files are closed after having been initially written. This feature can e.g. be used to automatically replicate volumes that only contain write-once files, such as archival data.

To configure the behavior of the on-close replication, the `xtfs_repl` tool is used.

The number of replicas to be created when a file is closed can be specified as a volume-wide parameter, which can be set as follows:

```
$> xtfs_repl --ocr_factor_set 2 /xtreemfs
```

This will automatically create a second replica when the file is closed, which implies that the file will be made read-only. Note that by setting the replication factor to 1 (default value), on-close replication will be switched off, which means that the file won't be replicated and will remain writable after having been closed.

The current replication factor of a volume can be retrieved as follows:

```
$> xtfs_repl --ocr_factor_get /xtreemfs
```

Moreover, it is possible to specify whether an automatically created replica will be synchronized in the background or on demand. By default, replicas will be synced on demand. This can be changed as follows:

```
$> xtfs_repl --ocr_full_set true /xtreemfs
```

Depending on whether `--ocr_full_set` is `true` or `false`, background replication of newly created files is switched on or off.

To show whether replicas are automatically filled or not, execute the following command:

```
$> xtfs_repl --ocr_full_get /xtreemfs
```

Changing OSD and Replica Selection Policies

When creating a new file, OSDs have to be selected on which to store the file content. Likewise, OSDs have to be selected for a newly added replica, as well as the order in which replicas are contacted when accessing a file. How these selections are done can be controlled by the user.

OSD and replica selection policies can only be set for the entire volume. Further details about the policies are described in Sec. 2.7.3.

The policies are set and modified with the `xtfs_repl` tool. A policy that controls the selection of a replica is set as follows:

```
$> xtfs_repl --rsp_set dcmap /xtreemfs
```

This will change the current replica selection policy to a policy based on a data center map. The current replica selection policy is shown as follows:

```
$> xtfs_repl --rsp_get /xtreemfs
```

Note that by default, there is no replica selection policy, which means that the client will attempt to access replicas in their natural order, i.e. the order in which the replicas have been created.

Similar to replica selection policies, OSD selection policies are set and retrieved:

```
$> xtfs_repl --osp_set dcmap /xtreemfs
```

sets a data center map-based OSD selection policy, which is invoked each time a new file or replica is created. The following predefined policies exist (see Sec. 2.7.3 and `man xtfs_repl` for details):

- `default`
- `fqdn`
- `dcmap`
- `vivaldi`

The default OSD selection policy selects a random subset of OSDs that are responsive and have more than 2GB of free disk space, whereas the `fqdn` and `dcmap` policies select those subsets of responsive OSDs with enough space that are closest according to fully qualified domain names and a data center map, accordingly. The `vivaldi` policy uses the vivaldi coordinates of OSDs and clients for selecting the closest replica. Besides, custom policies can be set by passing a list of basic policy IDs to be successively applied instead of a predefined policy name.

The OSD selection policy can be retrieved as follows:

```
$> xtfs_repl --osp_get /xtreemfs
```

Setting and Listing Policy Attributes

OSD and replica selection policy behavior can be further specified by means of policy attributes. For a list of predefined attributes, see `man xtfs_repl`. Policy attributes can be set as follows:

```
$> xtfs_repl --pol_attr_set domains "*.xtreemfs.org bla.com" \
   /xtreemfs
```

A list of all policy attributes that have been set can be shown as follows:

```
$> xtfs_repl --pol_attrs_get /xtreemfs
```

2.6.4 Vivaldi

Client machines that want to use vivaldi network coordinates for replica and OSD selection must calculate their own coordinates relative to the OSDs. This is done by the `xtfs_vivaldi` utility which must be started on each client machine. Ideally, this process is started during boot with the `xtreemfs-vivaldi` init.d scripts provided. The utility must be started with the directory service address and the path to a file in which the coordinates are stored.

```
$> xtfs_vivaldi remote.dir.machine \
   /var/lib/xtreemfs/vivaldi_coordinates
```

If started with the init.d script, the utility will get the DIR address from `/etc/xos/xtreemfs/default_dir` and will store the coordinates in `/var/lib/xtreemfs/vivaldi_coordinates`.

The coordinate file must be passed as an argument when mounting a volume:

```
$> mount.xtreemfs --vivaldi-coordinates-file-path /var/lib/xtreemfs/vivaldi_coordina
   remote.dir.machine/myVolume /xtreemfs
```

Finally, the vivaldi replica and OSD selection policies must be set at the MRC for the volume(s). See Sec. 2.6.3 for details.

2.7 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. Policies are a means to customize an XtreemFS installation.

XtreemFS supports a range of predefined policies for different tasks. Alternatively, administrators may define their own policies in order to adapt XtreemFS to customer demands. This chapter contains information about predefined policies, as well as mechanisms to implement and plug in custom policies.

2.7.1 Authentication Policies

Any operation on a file system is executed on behalf of a user. The process of determining the user bound to a request is generally referred to as *user authentication*. To render user authentication customizable, the MRC allows administrators to specify an authentication policy by means of an *Authentication Provider*. Authentication Providers are modules that implement different methods for retrieving user and group IDs from requests.

The following predefined authentication providers exist:

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 kinds of X.509 certificates 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 Organizational 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 (global UID and global GIDs), but the behavior is similar to the SimpleX509AuthProvider.

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

2.7.2 Authorization Policies

Before executing an operation, a file system needs to check whether the user bound to the operation is sufficiently authorized, i.e. is allowed to execute the operation. User authorization is managed by means of *access policies*, which reside on the MRC. Unlike authentication policies which are bound to an MRC, access policies can be defined for each volume. This has to be done when the volume is created (see `man xtfs_mkvol`). 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 evaluation of access rights is needed.
- 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.

2.7.3 OSD and Replica Selection Policies

When a new file is created or a replica is automatically added to a file, the MRC must decide on a set of OSDs for storing the file content. To select the most suitable subset among all known OSDs, OSD Selection Policies are used.

Replica selection is a related problem. When a client opens a file with more than one replica, the MRC uses a replica selection policy to sort the list of replicas for the client. Initially, a client will always attempt to access the first replica in the list received from the MRC. If a replica is not available, it will automatically attempt to access the next replica from the list, and restart with the first replica if all attempts have failed. Replica selection policies can be used to sort the replica lists, e.g. to ensure that clients first try to access replicas that are close to them.

Both OSD and replica selection policies share a common mechanism, in that they consist of *basic policies* that can be arbitrarily combined. Input parameters of a basic policy are a set of OSDs, the list of the current replica locations of the file, and the IP address of the client on behalf of whom the policy was called. The output parameter is a filtered and potentially sorted subset of OSDs. Since OSD lists returned by one basic policy can be used as input parameters by another one, basic policies can be chained to define more complex composite policies.

OSD and replica selection policies are assigned at volume granularity. For further details on how to set such policies, please refer to Sec. 2.6.3.

Attributes

The behavior of basic policies can be further refined by means of policy attributes. Policy attributes are extended attributes with a name starting with `xtreemfs.policies.`, such as `xtreemfs.policies.minFreeCapacity`. Each time a policy attribute is set, all policies will be notified about the change. How an attribute change affects the policy behavior depends on the policy implementation.

Predefined Policies

Each basic policy can be assigned to one of the three different categories called *filtering*, *grouping* and *sorting*. *Filtering policies* generate a sublist from a list of OSDs. The sublist only contains those OSDs from the original list that have a certain property. *Grouping policies* are used to select a subgroup from a given list of OSDs. They basically work in a similar manner as filtering policies, but unlike filtering policies, they always return a list of a fixed size. *Sorting policies* generate and return a reordered list from the input OSD list, without removing any OSDs.

The following predefined policies exist:

Filtering Policies

- **Default OSD filter (policy ID 1000)**
  Removes OSDs from the list that are either dead or do not have sufficient space. By default, the lower space limit for an OSD is 2GB, and the upper response time limit is 5 minutes.

  Attributes:

  - *free_capacity_bytes*: the lower space limit in bytes
  - *offline_time_secs*: the upper response time limit in seconds

- **FQDN-based filter (policy ID 1001)**
  Removes OSDs from the list that do not match any of the domains in a given set. By default, the set of domains contains '*', which indicates that no domains are removed.

  Attributes:

  - *domains*: a comma or space-separated list of domain names. The list may include leading and trailing '*'s, which will be regarded as wildcard characters.

Grouping Policies

- **Data center map-based grouping (policy ID 2000)**
  Removes all OSDs from the OSD set that have been used in the file's replica locations list already and selects the subset of OSDs that is closest to the client and provides enough OSDs for the new replica in a single data center.

  This policy uses a statically configured datacenter map that describes the distance between datacenters. It works only with IPv4 addresses at the moment. Each datacenter has a list of matching IP addresses and networks which is used to assign clients and OSDs to datacenters. Machines in the same datacenter have a distance of 0.

  This policy requires a datacenter map configuration file in `/etc/xos/xtreemfs/datacentermap` on the MRC machine which is loaded at MRC startup. This config file must contain the following parameters:

  - `datacenters=A,B,C`
    A comma separated list of datacenters. Datacenter names may only contain a-z, A-Z, 0-9 and _.

  - `distance.A-B=100`
    For each pair of datacenters, the distance must be specified. As distances are symmetric, it is sufficient to specify A to B.
  - `addresses.A=192.168.1.1,192.168.2.0/24`
    For each datacenter a list of matching IP addresses or networks must be specified.
  - `max_cache_size=1000`
    Sets the size of the address cache that is used to lookup IP-to-datacenter matches.

A sample datacenter map could look like this:

```
datacenters=BERLIN,LONDON,NEW_YORK
distance.BERLIN-LONDON=10
distance.BERLIN-NEW_YORK=140
distance.LONDON-NEW_YORK=110
addresses.BERLIN=192.168.1.0/24
addresses.LONDON=192.168.2.0/24
addresses.NEW_YORK=192.168.3.0/24,192.168.100.0/25
max_cache_size=100
```

- **FQDN-based grouping (policy ID 2001)**
  Removes all OSDs from the OSD set that have been used in the file's replica locations list already and selects the subset of OSDs that is closest to the client and provides enough OSDs for the new replica in a single domain.

  This policy uses domain names of clients and OSDs to determine the distance between a client and an OSD, as well as if OSDs are in the same domain.

Sorting Policies

- **Shuffling (policy ID 3000)**
  Shuffles the given list of OSDs.
- **Data center map-based sorting (policy ID 3001)**
  Sorts the list of OSDs in ascending order of their distance to the client, according to the data center map.

- **Vivaldi network coordinates based sorting (policy ID 3003)**
  Sorts the list of OSDs in ascending order of their distance to the client, according to the vivaldi coordinates of the client and OSDs. This policy requires the clients to run the `xtfs_vivaldi` service.

- **DNS based OSD Selection (policy ID 3002)**
  The FQDN of the client and all OSDs is compared and the maximum match (from the end of the FQDN) is used to sort the OSDs. The policy sorts the list of OSDs in descending order by the number of characters that match. This policy can be used to automatically select OSDs which are close to the client, if the length of the match between two DNS entries also indicate a low latency between two machines.

2.7.4 Striping Policies

XtreemFS allows the content, i.e. the objects 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 OSDs in order to increase throughput. To configure how files are striped, XtreemFS supports *striping policies*.

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. If not enough OSDs are available when the file is created, the number of available OSDs will be used instead; if it is 0, an I/O error is reported to the client. The *stripe size* defines the size of each object.

Striping over several OSDs enhances the read and write throughput to a file. The maximum throughput depends on the striping width. However, using `RAID0` also increases the probability of data loss. If a single OSD fails, parts of the file are no longer accessible, which generally renders the entire file useless. Replication can mitigate the problem but has all the restrictions described in Sec. 2.6.3.

2.7.5 Plug-in Policies

To further customize XtreemFS, the set of existing policies can be extended 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`
  interface for authentication policies

- `org.xtreemfs.mrc.ac.FileAccessPolicy`
  interface for file access policies

- `org.xtreemfs.mrc.osdselection.OSDSelectionPolicy`
  interface for OSD and replica selection policies

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. 2.4.2, 2.4.2), 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.

Chapter 3

The OSS Library Interface and User Guide

3.1 Overview

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/Intel64 compatible processors.

3.2 Changes and New Features

This section summarizes the changes and new features between OSS releases 0.4 and 0.5.

3.2.1 Transaction Management

OSS's transaction management now includes a new ultra-peer commit protocol. In comparison to the peer-to-peer commit protocol, ultra-peer commit achieves better performance in wide-area networks. The peer-to-peer and ultra-peer commit protocols have been consolidated, such that the commit protocol can be chosen during configuration of the library. The monitoring subsystem in OSS is now able to track object accesses in the peer-to-peer commit protocol, which is important for adaptive grouping of object accesses and for adaptive replication.

Aside from commit-related functionality, the stack unwinding mechamism has been modified to allow restarting applications that have been compiled with strong optimization. To improve scalability of applications that use OSS on modern multiprocessor computers, OSS now supports multi-threading in transactional applications.

3.2.2 Communication

The performance of several communication layers has been improved during development of OSS 0.5. The transfer of Millipages now is much more efficient. The degree of parallelization in the networking and communication subsystem has been enhanced. OSS can now receive messages from many nodes concurrently, such that fragmented messages or temporary node failures do not stall the communication subsystem any more. OSS 0.5 enables NAT compatibility for outgoing connections, which is important when sharing objects over wide-area networks.

To facilitate the synchronization of distributed applications, OSS 0.5 provides an implementation of distributed barriers.

Furthermore, OSS now supports the dynamic joining of nodes. In order to help debug inter-node communication, we have developed a packet dissector for OSS.

3.3 Installation of OSS

You can install OSS either using the prebuild distribution packages which are 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.

3.3.1 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 release* to install the packaged version of OSS (Library and applications). If you have XtreemOS already installed and wish to install OSS, select it in the package management dialog, or run the following commands as root:

```
$> urpmi liboss0
$> urpmi oss
```

The first command installs the OSS library, the second command installs some example applications.

Application development based on OSS need the following additional packages:

```
$> urpmi liboss0-devel
$> urpmi liboss0-static-devel
```

3.3.2 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. The OSS sources can be installed from XtreemOS source repository:

```
$> urpmi oss-0.5.1-1xos2.0.src    (OSS sources)
```

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.

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

- libglib2.0-devel $\geq$ 2.18
- libreadline5-devel

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. The following step allows altering the default configuration of OSS (hardware architecture, features, ...) if desired. If this step is omitted, OSS will be built in its default configuration. A description of the configuration options can be found in Appendix B.

```
$> make menuconfig
```

The following command builds the OSS library:

```
$> 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 pass configuration parameters via command line or to enable non-standard features, you can directly supply the corresponding parameters to make. For example, run `make -B ARCH=I686` to build OSS for 32-Bit x86 machines and `make -B ARCH=X86_64` to build OSS for 64-Bit x86 machines respectively.

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.

3.3.3 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 according to the example below:

```
Object Sharing Service version 0.5.1 architecture I686
   subversion revision 5844 (2010-03-05 14:51:38)
build 1
Object Sharing Service has been installed correctly.
```

Simple test of Object Sharing

The simple test application (*oss_simple*) 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 transaction boundaries
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* is very simple with one sphere in the center, a few bowls around and only a few lights. *Scene2* is very complex with some arrangements of bowls and reflecting walls. *Scene3* displays the letters ”‘OSS”’ consisting of bowls. You can write own scenes as *C* files analoguous to *SceneDemo1.c*.

3.4 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 contrast, WP3.1 has defined an XOSAGA interface for object sharing that represents the OSS interface in a portable way.

3.4.1 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.

```
void *
oss_nameservice_get(
     const char *id
     );

void
oss_nameservice_set(
     const char *id,
     void *val
     );
```

OSS contains a simple name service, which applications can use to store and retrieve object IDs. The name service has a tree structure, with slashes (/) separating directory levels. Each entry begins with a slash. An application or OSS module can set a value for a name by calling `oss_nameservice_set` and retrieve a value by calling `oss_nameservice_get`. A value that has not yet been set is treated as object ID `NULL`.

```
void
oss_wait(
     void *addr,
     unsigned char value
     );
```

The barriers implementation allows applications to wait until the character object pointed to by `addr` contains a target value. The character object may be subject to transactional or strong consistency.

3.4.2 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>`.

3.5 Performance Measurements

Figure 3.1 shows performance measurements of the transactional memory provided by OSS. The results do not illustrate the overall performance of OSS rather the conflict and network related penalties for a best and worst case scenario. Therefore, measurements have been done with all optimizations (local commits, linked transactions, consistency domains etc.) turned off. In the worst case scenario all peers concurrently increment a common variable stored in the transactional memory, in the best case scenario each peer increments its own variable. The best case scenario (private variable) solely shows the network overhead produced by the commit protocol. The worst case additionally causes penalties due to a high conflict rate which results in transaction aborts and restarts.

For the measurements we have used our P2P commit protocol with two different token mechanism for transaction serialization. The token was passed among the peers either by a dedicated coordinator or P2P based approach. Furthermore, we have expanded the transaction duration and pause between to successive transactions to simulate real live applications. The red and green lines show the overall transaction thoughput by using the coordinated and p2p based token passing mechanism. The black line shows the maximum theoretical throughput based on the transaction time and pause, presumed no conflicts occur.

The diagrams in the left column show the negative impact of network communication, but nevertheless growing of overall transaction throughput. To improve OSS' performance the implemented optimizations aim at exploiting locality to prevent unnecessary network communication (e. g. local commits) and linked transactions to hide the network latency by acquiring the token in the background while starting the next transaction. The right column shows, that the programmer must be aware of transactional conflicts and performance issues. So he has to optimize its program to get a low conflict rate.

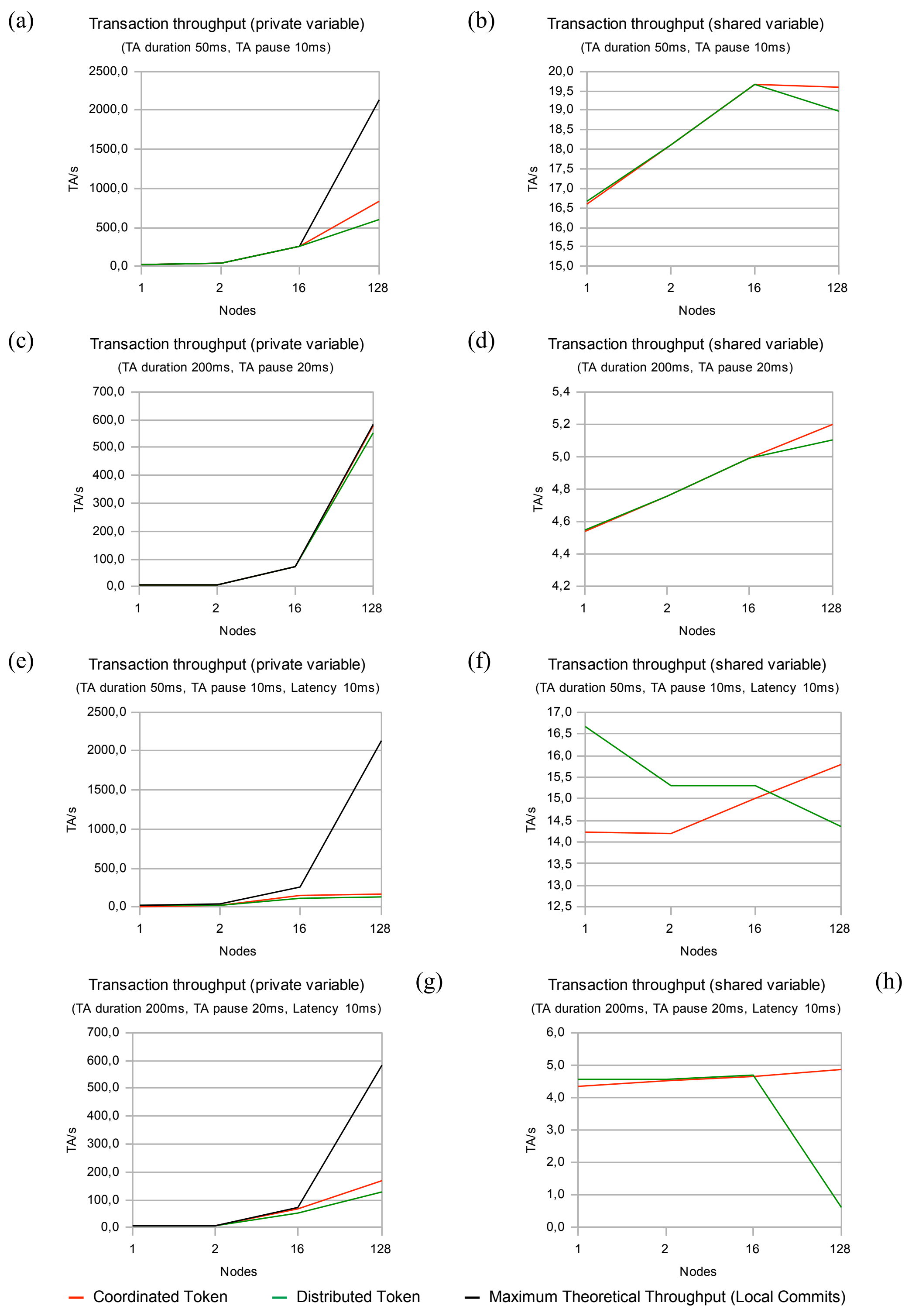


Figure 3.1: Performance measurements of conflicting and non-conflicting variable incrementations

Appendix A

XtreemFS Appendix

A.1 Support

Please visit the XtreemFS website at www.xtreemfs.org for links to the user mailing list, bug tracker and further information.

A.2 XtreemOS Integration

A.2.1 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 the 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
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. Make sure that all paths and passphrases (`xtreemfs` in this example) are correct.

A.3 Hadoop Integration

A.3.1 Introduction

XtreemFS is a distributed filesystem that can be used instead of HDFS the distributed filesystem made by the developers of Hadoop.

Therefore it replaces the NameNode and the Datanodes provided by HDFS in a common Hadoop setup. A DIR is used instead of a NameNode, because it stores the information about where the files and there metadata are located at the OSDs and the MRC, like the NameNode does for DataNodes. These DataNodes hold the files that have been stored at HDFS. On XtreemFS these files are split into metadata and raw filedata to be stored seperated at a MRC and OSDs.

The three master services JobTracker, DIR and MRC are required in a Hadoop configuration. They can run alone or in arbitrary combinations on the same machine. Hadoop can be used with an arbitrary number of Slaves. It is recommended to run a TaskTracker together with an OSD on each Slave machine to improve performance, but it is not mandatory.

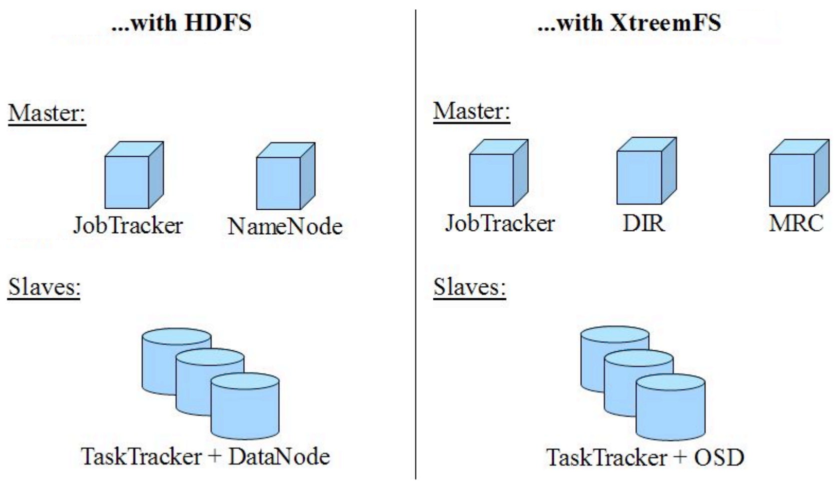


Figure A.1: Hadoop cluster setup recommendation

A.3.2 Quick Start

This section will help you to set up a simple Hadoop configuration with all necessary services running on the same host.

Required software:

- XtreemFS servers (v 1.2.1) including XtreemFS.jar and yidl.jar (www.XtreemFS.org)
- HadoopClient.jar (www.XtreemFS.org)
- Hadoop (v 0.20.1) (hadoop.apache.org)
- JDK 1.6+ (Oracle/SUN)

Setup:

1. Install and start XtreemFS:
   Follow the instructions given by the quick start guide for XtreemFS, available at Sec.2.2. Notice that the DIR is reachable at *localhost:32638*, beause this information will be important later.
2. Download and extract Hadoop
3. Configure Hadoop to use XtreemFS instead of HDFS:

(a) After downloading and extracting Hadoop you first have to add XtreemFS, the HadoopClient and yidl to its classpath. To do so edit the *hadoop-env.sh* that can be found in the *conf* directory of Hadoop and add the paths to XtreemFS.jar, yidl.jar and HadoopClient.jar separated by '*:*' to the HADOOP_CLASSPATH. If you run a Linux-based OS these jar-libraries are located at '*/usr/share/java/*'.

(b) Now you have to specify some properties at the *core-site.xml* which also has to be in the *conf* directory of Hadoop. If this file does not exist you can safely create it.

```
<configuration>

<property>
  <name>fs.xtreemfs.impl</name>
  <value>org.xtreemfs.common.clients.hadoop.XtreemFSFileSystem</value>
  <description>The FileSystem for xtreemfs: uris.</description>
</property>

<property>
  <name>fs.default.name</name>
  <value>xtreemfs://localhost:32638</value>
  <description>Address for the DIR.</description>
</property>

<property>
  <name>xtreemfs.volumeName</name>
  <value>volumeName</value>
  <description>Name of the volume to use within XtreemFS.</description>
</property>

</configuration>
```

i. The first property is required to register the HadoopClient of XtreemFS at Hadoop. Now you are able to access XtreemFS by the Hadoop binary using the *fs* argument.

ii. The next property makes Hadoop use the DIR instead of a NameNode, therefore address and port of the DIR has to be populated. In this case the DIR is located at *localhost:32638.*

iii. The last property specifies the name of the volume to use within XtreemFS. Make sure, that the volume (here named *volumeName*) does exist. If the volume is not available Hadoop will not be able to use XtreemFS!

Hint: If you want to provide userrights to your Hadoop installation according to the POSIX file-access-policy, you have to set the following additional properties:

```
<property>
  <name>xtreemfs.client.userid</name>
  <value>hadoopUserID</value>
  <description>UserID to be used by Hadoop while accessing XtreemFS.
</property>

<property>
  <name>xtreemfs.client.groupid</name>
  <value>hadoopGroupID</value>
  <description>GroupID to be used by Hadoop while accessing XtreemFS
</property>
```

4. To provide the minimum JobTracker configuration for Hadoop you have also to add the following property to the *conf/mapred-site.xml*:

```
<configuration>

<property>
  <name>mapred.job.tracker</name>
  <value>localhost:9001</value>
  <description>Listening address for the JobTracker.</description>
</property>

</configuration>
```

   Which specifies the address where the JobTracker will be running at.

5. Finally you are now able to start the JobTracker by running '*bin/hadoop jobtracker*' from within the Hadoop root-directory and a TaskTracker by executing '*bin/hadoop tasktracker*'.

Congratulations! You successfully finished the quick start guide of the XtreemFS-Hadoop integration and are now able to use your Hadoop applications like as is well known or go on with the tutorials available on hadoop.apache.org.

A.4 Command Line Utilities

xtfs_cleanup Deletes orphaned objects on an OSD and restores orphaned files.

lsfs.xtreemfs (was xtfs_lsvol) Lists the volumes on an MRC.

mkfs.xtreemfs (was xtfs_mkvol) Creates a new volume on an MRC.

mount.xtreemfs (was xtfs_mount) The XtreemFS client which mounts an XtreemFS volume locally on a machine.

xtfs_mrcdbtool Dumps and restores an XML representation of the MRC database.

xtfs_repl Controls file replication in XtreemFS.

rmfs.xtreemfs (was xtfs_rmvol) Deletes a volume.

xtfs_sp Displays and modifies default striping policies for directories and volumes.

xtfs_scrub Examines all files in a volume for wrong file sizes and checksums and corrects wrong file sizes in the MRC.

xtfs_stat Displays XtreemFS-specific file information, such as OSD lists and striping policies.

xtfs_test Automatically sets up an XtreemFS testing environment and runs the automatic XtreemFS test suite.

umount.xtreemfs (was xtfs_umount) Un-mounts a mounted XtreemFS volume.

xtfs_vivaldi client service to calculate vivaldi coordinates.

Appendix B

OSS Appendix

B.1 Support

Please visit the OSS website at the University of Duesseldorf to contact developers and further information on OSS. The XtreemOS bugtracker is available at SourceForge.

B.2 OSS Configuration Options

This chapter describes all configuration options of OSS, which affect compilation of OSS. The configuration dialog is accessible via

```
$> make menuconfig
```

The configuration dialog is modelled after the Linux kernel configuration dialog. Press the *enter* key to select an item and press the *space* key to toggle a selection. Use the cursor keys to navigate between items, to exit from a menu or to display a help text for the selected item.

B.2.1 Debugging

Library developers can configure a number of debugging options.

debug level for whole build process

Selects a global level for debug output unless this value is overridden by a *per-file debug level.*

debug glib

Enables debug output for glib related operations.

debug networking

Enables debug output for network related operations.

Per-file debug levels

Allows a fine granular debug level selection for specific source files.

B.2.2 Code generation

The binary code of the OSS library can be compiled for different processor architectures.

Processor Architecture

Defines the processor architecture for which OSS is compiled. OSS supports the following architectures:

- AMD64/Intel64 architecture (64-bit operating system provided)
- I686 architecture (32-bit or 64-bit supported)

The usage of a 32-bit OSS version in an 64-bit XtreemOS system requires the installation of a 32-bit compatibility layer (32-bit libraries).

B.2.3 Library Interface

In addition to the base functions which the OSS library always exports, a number of functions are tagged as optional or experimental.

oss_mmap

Exports the command `oss_mmap` which creates an object from the content of a file, and the commands `oss_munmap` and `oss_msync`, which will unmap and synchronize object and file in a future version of OSS.

oss_sync/oss_push/oss_pull

Provides three additional calls for explicit synchronization of weakly consistent objects. These calls have not yet been implemented in the current OSS release.

oss_nameservice_get/oss_nameservice_set

Exports the functions of the nameservice to the API. This allows applications to use the internal nameservice of OSS.

nameservice consistency

Selects the consistency model of nameservice entries. Some internally defined entries are always handled according to strong consistency.

miscellaneous debug functions

Exports further debugging functions to the API *(see oss.h)*.

unstable library interface

Exports functions for retrieving the own node id, the number of nodes, and setting the number of nodes participating on transactional consistency to the API *(see oss.h)*. These functions are used for debugging purposes only. *(Without claim to be still available in future versions of OSS)*.

oss_wait

Enables distributed barriers for strong and transactional consistency.

hashmap

Exports the functions for managing a hashmap of shared objects. This allows applications to use the internal hashmap implementation of OSS.

B.2.4 Communication

The OSS library interface deliberately does not specify how nodes are interconnected. Internal to the OSS library, node interconnection can be implemented in several ways.

Overlay Routing

Allows configuration of overlay network related options. Unless selected, the node network is fully meshed; however, connections are established on demand.

Superpeer Network

Enables routing of OSS messages in the overlay network [**Experimental**].

B.2.5 Monitoring

For performance measurements as well as for automatic reconfiguration during runtime, the library contains a monitoring subsystem. The subsystem allows the library developer to intersperse monitoring events in the source code. Different handlers can be attached to monitoring events by specifying their names in the configuration dialog. The default no-op handler is called `null`. The `count` handler simply counts the number of events. The `printf` handler prints the events seen immediately, including the source code location and a custom pointer value. The `latency` handler measures the duration of events, whereas the `slist` handler accumulates the pointer values of all events seen in a singly-linked list.

monitoring

Enables monitoring of several OSS internal operations for statistics and dynamic reconfiguration.

log monitor data to file

Enables logging the monitoring data to a file. Unless selected, the monitors print statistics to standard output.

periodic dump

Time interval in seconds of periodic monitoring data dump.

short log

Reduces verbosity of logging information output.

clock

Selects the time source for the monitoring subsystem. Use `clock_gettime` to measure elapsed time in micro-seconds, use `rdtsc` to measure CPU clock cycles, or use `gettimeofday` to measure elapsed time in micro-seconds using a POSIX-compliant call.

object_mmap

Monitors object mappings.

object_alloc

Monitors object allocations.

object_free

Monitors object deallocations.

read_fault

Monitors detected read accesses.

write_fault

Monitors detected write accesses.

read_access

Monitors read accesses evoked by a test application that has been prepared to announce read accesses.

write_access

Monitors write accesses evoked by a test application that has been prepared to announce write accesses.

B.2.6 Memory allocator

OSS supports different memory allocators.

mspace allocation from dlmalloc

Enables the mspaces memory allocator. The mspace allocator is a general-purpose allocator, which is very reliable and versatile.

millipage implementation

Enables the millipage memory allocator. This allocator concentrates multiple objects allocated on different memory pages on one physical page frame. The millipage allocator is well suited for allocations of small objects if another allocator might induce false sharing.

simple list allocator

Enables the simple first-fit memory allocator. The simple list allocator is very fast for allocations, but frequent deallocations may induce external fragmentation.

replica management

Currently, replication is handled using invalidations and requests for invalid objects. A future release of the library will include a full-featured replica management that handles a combination of object invalidations and updates.

diff computation and transfer

Diff computation and transfer will speed up object accesses, but it is still under development and not included in the current release.

B.2.7 Applications

build raytracer

Builds the raytracer application, shipped with OSS.

build wissenheim

Builds the wissenheim application out of OSS. This option is only intended for debugging purposes regarding Wissenheim over OSS. Wissenheim on XtreemOS comes with its own build system.

B.2.8 Remote installation

UDUS infrastructure specific options *(not for public usage).*

Index