
# The XtreemFS Installation and User Guide

Version 1.3

XtreemFS is available from the XtreemFS website (www.XtreemFS.org).

# Contents


1 Quick Start xi

2 About XtreemFS 1

2.1 What is XtreemFS? . . . 1

What makes XtreemFS a distributed file system? . . . 1

What makes XtreemFS a replicated file system? . . . 1

2.2 Is XtreemFS suitable for me? . . . 2

XtreemFS is ... . . . 2

XtreemFS is not ... . . . 2

2.3 Core Features . . . 3

Distribution. . . . 3

Replication. . . . 3

Striping. . . . 3

Security. . . . 4

2.4 Architecture . . . 4

XtreemFS Components. . . . 4

3 XtreemFS Services 7

3.1 Installation . . . 7

3.1.1 Prerequisites . . . 7

3.1.2 Installing from Pre-Packaged Releases . . . 7

3.1.3 Installing from Sources . . . 8

3.2 Configuration . . . 8

3.2.1 A Word about UUIDs . . . 9

3.2.2 Automatic DIR Discovery . . . 9

3.2.3 Authentication . . . 9

3.2.4 Configuring SSL Support . . . 10

Converting PEM files to PKCS#12 . . . 10

Importing trusted certificates from PEM into a JKS . . . . . 11
Sample Setup . . . . . 11
3.2.5 List of Configuration Options . . . . . 13
admin_password *optional* . . . . . 13
authentication_provider . . . . . 13
babudb.baseDir . . . . . 14
babudb.cfgFile *optional* . . . . . 14
babudb.checkInterval *optional* . . . . . 14
babudb.compression *optional* . . . . . 14
babudb.debug.level *optional* . . . . . 15
babudb.logDir . . . . . 15
babudb.maxLogfileSize *optional* . . . . . 15
babudb.pseudoSyncWait *optional* . . . . . 16
babudb.sync . . . . . 17
babudb.worker.maxQueueLength *optional* . . . . . 17
babudb.worker.numThreads *optional* . . . . . 18
capability_secret . . . . . 18
capability_timeout *optional* . . . . . 18
checksums.enabled . . . . . 18
checksums.algorithm . . . . . 18
debug.level *optional* . . . . . 19
debug.categories *optional* . . . . . 20
dir_service.host . . . . . 20
dir_service.port . . . . . 20
discover *optional* . . . . . 21
enable_local_FIFOs *optional* . . . . . 21
flease.dmax_ms *optional* . . . . . 21
flease.lease_timeout_ms *optional* . . . . . 21
flease.message_to_ms *optional* . . . . . 22
flease.retries *optional* . . . . . 22
geographic_coordinates *optional* . . . . . 22
hostname *optional* . . . . . 22
http_port . . . . . 22
ignore_capabilities *optional* . . . . . 23
listen.address *optional* . . . . . 23
listen.port . . . . . 23
local_clock_renewal . . . . . 23

monitoring.enabled . . . . . 24
monitoring.email.programm . . . . . 24
monitoring.email.receiver . . . . . 24
monitoring.email.sender . . . . . 24
monitoring.max_warnings . . . . . 24
monitoring.service_timeout_s . . . . . 25
no_atime . . . . . 25
object_dir . . . . . 25
osd_check_interval . . . . . 25
policy_dir *optional* . . . . . 25
remote_time_sync . . . . . 26
renew_to_caps *optional* . . . . . 26
report_free_space . . . . . 26
socket.send_buffer_size *optional* . . . . . 26
socket.recv_buffer_size *optional* . . . . . 26
ssl.enabled . . . . . 27
ssl.grid_ssl . . . . . 27
ssl.service_creds . . . . . 27
ssl.service_creds.container . . . . . 27
ssl.service_creds.pw . . . . . 28
ssl.trusted_certs . . . . . 28
ssl.trusted_certs.container . . . . . 28
ssl.trust_manager *optional* . . . . . 28
ssl.trusted_certs.pw . . . . . 28
startup.wait_for_dir . . . . . 28
storage_layout *optional, experimental* . . . . . 29
uuid . . . . . 29

3.3 Execution and Monitoring . . . . . 29
3.3.1 Starting and Stopping the XtreemFS services . . . . . 29
3.3.2 Web-based Status Page . . . . . 29
3.3.3 DIR Service Monitoring . . . . . 30
3.4 Troubleshooting . . . . . 30

4 **XtreemFS Client** 33
4.1 Installation . . . 33
4.1.1 Prerequisites . . . 33
4.1.2 Installing from Pre-Packaged Releases . . . 33
4.1.3 Installing from Sources . . . 34
4.2 Volume Management . . . 34
4.2.1 Creating Volumes . . . 35
4.2.2 Deleting Volumes . . . 35
4.2.3 Listing all Volumes . . . 35
4.3 Accessing Volumes . . . 36
4.3.1 Mounting and Un-mounting . . . 36
4.3.2 Mount Options . . . 36
4.4 Troubleshooting . . . 37

5 **XtreemFS Tools** 39
5.1 Installation . . . 39
5.1.1 Prerequisites . . . 39
5.1.2 Installing from Pre-Packaged Releases . . . 39
5.1.3 Installing from Sources . . . 40
5.2 Admin Tools . . . 40
5.2.1 MRC Database Conversion . . . 41
5.2.2 Scrubbing and Cleanup . . . 41
5.2.3 Setting the Service Status . . . 42
5.2.4 Snapshots . . . 43
5.3 User Tools . . . 44
5.3.1 `xtfsutil` for Files . . . 44
Changing the Replication Policy . . . 45
Adding and Removing Replicas . . . 45
5.3.2 `xtfsutil` for Volumes . . . 46
Changing the Default Striping Policies . . . 46
Changing the Default Replication Policy . . . 47
5.3.3 Changing OSD and Replica Selection Policies . . . 47
5.3.4 Setting and Listing Policy Attributes . . . 48
5.3.5 Modifying Access Control Lists . . . 48
5.4 Vivaldi . . . 49
5.5 Test Tools . . . 49

**6 Replication** 51
6.1 Read/Write File Replication . . . . . 51
6.2 Read-Only File Replication . . . . . 52
6.3 BabuDB Database Replication (DIR/MRC) . . . . . 52

**7 Policies** 53
7.1 Authentication Policies . . . . . 53
7.1.1 UNIX uid/gid - NullAuthProvider . . . . . 53
7.1.2 Plain SSL Certificates - SimpleX509AuthProvider . . . . . 54
7.2 Authorization Policies . . . . . 54
7.3 OSD and Replica Selection Policies . . . . . 55
7.3.1 Attributes . . . . . 55
7.3.2 Predefined Policies . . . . . 55
Filtering Policies . . . . . 55
Grouping Policies . . . . . 56
Sorting Policies . . . . . 57
7.4 Striping Policies . . . . . 57
7.5 Plug-in Policies . . . . . 58

**A Support** 59

**B Hadoop Integration** 61
B.1 Introduction . . . . . 61
B.2 Quick Start . . . . . 62

**C Command Line Utilities** 65

# Changes

Summary of important changes in release 1.3:

- **new client**
  We have re-written the client from scratch. The new client supports automatic failover for replicated files and metadata caching.
- **libxtreemfs**
  libxtreemfs is a convenient C++ library to use XtreemFS directly without a mounted client or the VFS layer. The new client is built on top of this library. A java version of libxtreemfs is planned.
- **File system snapshots**
  XtreemFS now supports snapshots. A snapshot reflects a momentary state of a volume or directory. It can be mounted and read-only accessed.
- **Full file replication**
  Starting with this release, XtreemFS supports full file replication. Read/write replicated files offer regular file system semantics and work with all applications.
- **DIR, MRC replication**
  The DIR and MRC can now be replicated using the BabuDB database replication. The replication works with a primary and backups. If the primary fails, a backup will automatically take over after a short time.
- **xtfsutil**
  We have replaced all user tools with a single binary. The new tool doesn't require java anymore.
- **OSD drain**
  With OSD drain, files can be removed from an OSD without interrupting the system. A fully drained OSD can be removed from the system without data loss.

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 srubber 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.

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

- **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. C.

- **"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. 3.2.4.

- **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.

- **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. 5.4.

# Chapter 1

# Quick Start

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

1. Download XtreemFS RPMs/DEBs and install
   (a) Download the RPMs or DEBs for your system from the XtreemFS website (http://www.xtreemfs.org)
   (b) open a root console (`su` or `sudo`)
   (c) install with `rpm -Uhv xtreemfs*-1.3.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.

# Chapter 2

# 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.

## 2.1 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.

## 2.2 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 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 Google's Protocol Buffers 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.** Starting with release 1.3, XtreemFS supports the replication of mutable files as well as a replicated Directory Service (DIR) and Metadata Catalog (MRC). All components in XtreemFS can be replicated for redundancy which results in a fully fault-tolerant file system. The replication in XtreemFS works with hot backups, which automatically take over if the primary replica fails.

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.

**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.

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.

## 2.4 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 the 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.

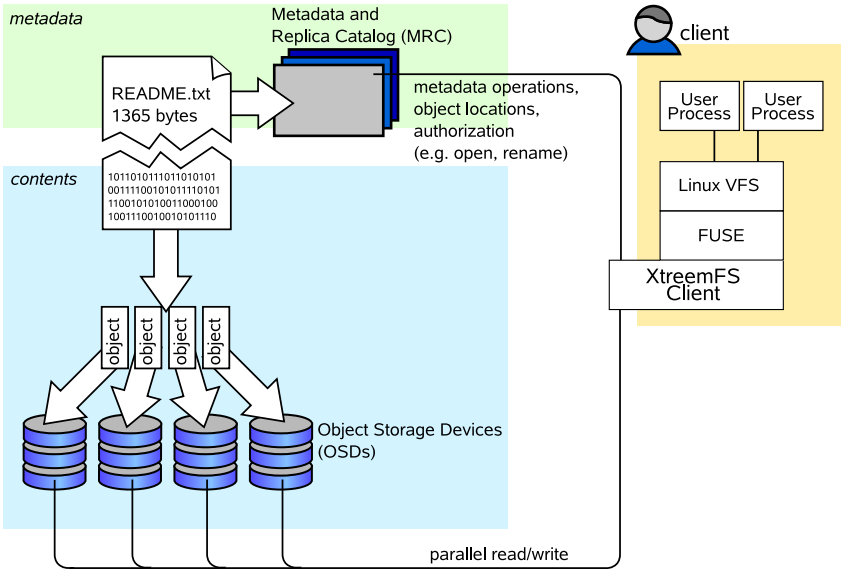



Figure 2.1: The XtreemFS architecture and components.

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.

# Chapter 3

# XtreemFS Services

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

## 3.1 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.

### 3.1.1 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.

### 3.1.2 Installing from Pre-Packaged Releases

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

```
$> rpm -i xtreemfs-server-1.3.x.rpm xtreemfs-backend-1.3.x.rpm
```

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

```
$> dpkg -i xtreemfs-server-1.3.x.deb xtreemfs-backend-1.3.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-1.3.x.rpm \
   xtreemfs-backend-1.3.x.rpm
```

on RPM-based distributions,

```
$> dpkg -i --ignore-depends java6-runtime \
   xtreemfs-server-1.3.x.deb xtreemfs-backend-1.3.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).

### 3.1.3 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.

## 3.2 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/`.

### 3.2.1 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.

### 3.2.2 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. 3.2.5, `dir_service`). By default. the DIR responds to UDP broadcasts. To disable this feature, set `discover = false` in the DIR service config file.

### 3.2.3 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. 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

### 3.2.4 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 certificates 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 `pbrpcg://` 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. 3.2 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. 3.3.1).

6. Create a new volume (see Sec. 4.2.1 for details).
Use

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

for SSL-enabled servers, or

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

for Grid-SSL-enabled servers.

7. Mount the volume (see Sec. 4.3 for details).

   Use

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

   for SSL-enabled servers, or

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

   for Grid-SSL-enabled servers.

### 3.2.5 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. 3.2.3 for details. |

`babudb.baseDir`

| | |
|---|---|
| Services | DIR, MRC |
| Values | absolute file system path to a directory |
| Default | DIR: `/var/lib/xtreemfs/dir/database`<br>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`<br>MRC: `config.db` |
| Description | Name for the database configuration file. |

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

| | |
|---|---|
| Services | DIR, MRC |
| Values | a positive integer value |
| Default | DIR: `300`<br>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`<br>MRC: `false` |
| Description | Flag that determines whether database content shall be compressed or not. |

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

| | |
|---|---|
| Services | DIR, MRC |
| Values | 0, 1, 2, 3, 4, 5, 6, 7 |
| Default | DIR: 4<br>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:<br>0 - fatal errors<br>1 - alert messages<br>2 - critical errors<br>3 - normal errors<br>4 - warnings<br>5 - notices<br>6 - info messages<br>7 - debug messages |

`babudb.logDir`

| | |
|---|---|
| Services | DIR, MRC |
| Values | absolute file system path |
| Default | DIR: `/var/lib/xtreemfs/dir/db-log`<br>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`<br>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`<br>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.sync`

| | |
|---|---|
| Services | DIR, MRC |
| Values | ASYNC, SYNC_WRITE_METADATA, SYNC_WRITE, FDATASYNC or FSYNC |
| Default | DIR: `FSYNC`<br>MRC: `ASYNC` |
| Description | The sync mode influences how operations are commited to the disk log before the operation is acknowledged to the caller.<br>- 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.<br>- 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.<br>- 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.<br>- FDATASYNC is similar to SYNC_WRITE but opens the file in asynchronous mode and calls fdatasync() after writing the data to disk.<br>- FSYNC is similar to SYNC_WRITE_METADATA but opens the file in asynchronous mode and calls fsync() after writing the data to disk.<br>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<br>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`<br>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. 3.2.2). (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. 3.2.2. |

`enable_local_FIFOs` *optional*

| | |
|---|---|
| Services | MRC |
| Values | true, false |
| Default | false |
| Description | Enables support for FIFOs (names pipes) on the local machine for compatibility reasons. If set to `false`, any attempt to open a FIFO will be rejected with EIO. Even if set to `true`, FIFOs will not work across multiple mounts. |

`flease.dmax_ms` *optional*

| | |
|---|---|
| Services | OSD |
| Values | millisconds |
| Default | 1000 |
| Description | Maximum clock drift between any two clocks in the system. If the actual drift between two server clocks exceeds this value, read-write replication may lead to inconsistent replicas. Since servers automatically synchronize their clocks with the clock on the DIR, however, the default 1000ms should be enough in most cases. |

`flease.lease_timeout_ms` *optional*

| | |
|---|---|
| Services | OSD |
| Values | millisconds |
| Default | 15000 |
| Description | Duration of a lease in milliseconds. For read-write-replicated files, the lease timeout specifies the validity time span of a master lease. Shorter lease timeouts guarantee a shorter failover period in the event of a server crash, which however comes at the cost of an increased rate of lease negotiations for each open file. The leaes timeout should be set to a value at least three times `flease.message_to_ms`. |

`flease.message_to_ms` ***optional***

| | |
|---|---|
| Services | OSD |
| Values | milliseconds |
| Default | 500 |
| Description | Time to wait for responses from other OSDs when negotiating leases for replicated files. This value should be larger than the maximum message round-trip time via TCP between any pair of OSDs. |

`flease.retries` ***optional***

| | |
|---|---|
| Services | OSD |
| Values | 1..1000 |
| Default | 3 |
| Description | Number of times to retry acquiring a lease for a replicated file before an IO error is sent to the client. |

`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. |

`ignore_capabilities` ***optional***

| | |
|---|---|
| Services | OSD |
| Values | true, false |
| Default | false |
| Description | When set to *true*, capability checks on the OSD are disabled. This property should only be set to *true* for debugging purposes, as it effectively overrides any security mechanisms on the system. |

`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,<br>MRC: 32636,<br>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.enabled**

| | |
|---|---|
| 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 `monitoring.email.receiver`, `monitoring.email.program`. In addition, you may want to change the values for `monitoring.email.sender`, `monitoring.max_warnings`, `monitoring.service_timeout_s`. |

**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` parameters. |

**monitoring.email.receiver**

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

**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` parameters. |

**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` parameters. |

`monitoring.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` parameters. |

`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. 7.3). This parameter defines the interval between two updates of the list of suitable OSDs. |

`policy_dir` ***optional***

| | |
|---|---|
| Services | MRC, OSD, DIR |
| Values | absolute file system path to a directory |
| Default | |
| Description | Directory containing user-defined policies and modules. When starting a service, the policy directory will be searched for custom policies. For further details on pluggable policies, see chapter 7. |

`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. |

`renew_to_caps` *optional*

| | |
|---|---|
| Services | MRC |
| Values | true, false |
| Default | false |
| Description | If set to true, the MRC allows capabilities to be renewed after they timed out. This parameter should only be used for debugging purposes, as it effectively overrides the revocation of access rights on a file. |

`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. 7.3). |

`socket.send_buffer_size` *optional*

| | |
|---|---|
| Services | OSD |
| Values | size in bytes |
| Default | -1 |
| Description | The send buffer size in bytes for sockets. -1 indicates that the default value (typically 128k) is used. |

`socket.recv_buffer_size` *optional*

| | |
|---|---|
| Services | OSD |
| Values | size in bytes |
| Default | -1 |
| Description | The receive buffer size in bytes for sockets. -1 indicates that the default value (typically 128k) is used. |

`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 `pbrpcg://` scheme prefix. |

`ssl.service_creds`

| | |
|---|---|
| Services | DIR, MRC, OSD |
| Values | path to file |
| Default | DIR: `/etc/xos/xtreemfs/truststore/certs/ds.p12`,<br>MRC: `/etc/xos/xtreemfs/truststore/certs/mrc.p12`,<br>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.trust_manager** ***optional***

| | |
|---|---|
| Services | DIR, MRC, OSD |
| Values | Java class name |
| Default | |
| Description | Sets a custom trust manager class for SSL connections. The trust manager is responsible for checking certificates when SSL connections are established. |

**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. |

`storage_layout` ***optional, experimental***

| | |
|---|---|
| Services | OSD |
| Values | HashStorageLayout |
| Default | HashStorageLayout |
| Description | Adjusts the internally used storage layout on the OSD. The storage layout determines how an OSD stores its files and objects. Currently, only `HashStorageLayout` is supported. |

`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`. |

## 3.3 Execution and Monitoring

This section describes how to execute and monitor XtreemFS services.

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

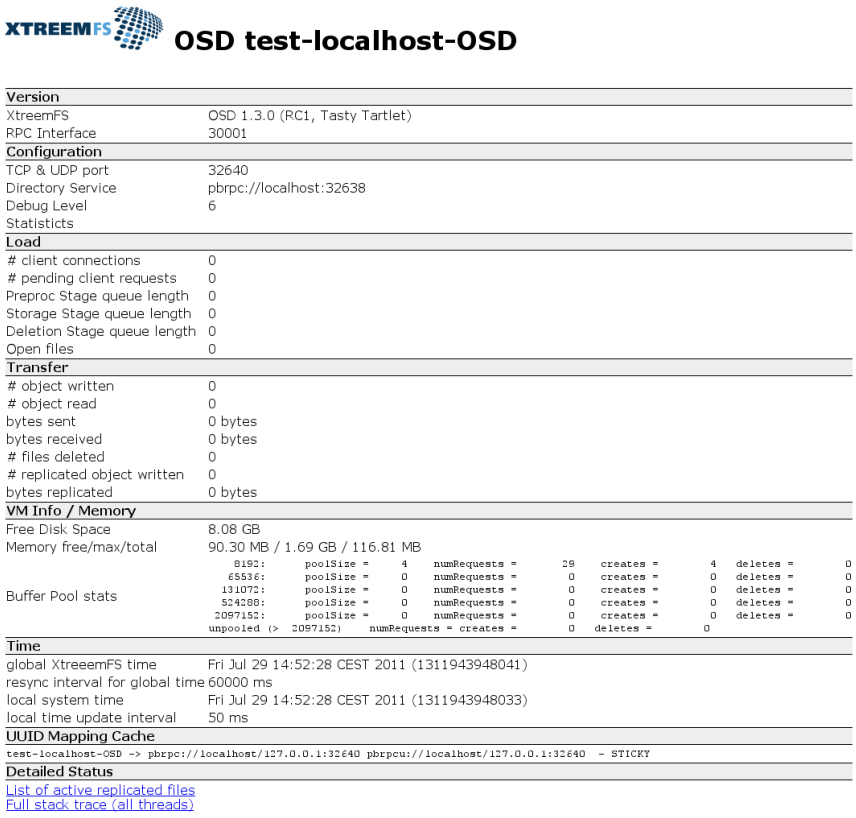
or

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

To run init.d scripts, root permissions are required. Note that MRC and OSD will wait for the Directory Service to become available before they start up. Once a Directory Service as well as at least one OSD and MRC are running, XtreemFS is operational.

### 3.3.2 Web-based Status Page

Each XtreemFS service can generate an HTML status page, which displays runtime information about the service (Fig. 3.1). 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.



XTREEMFS **OSD test-localhost-OSD**

| | |
|---|---|
| **Version** | |
| XtreemFS | OSD 1.3.0 (RC1, Tasty Tartlet) |
| RPC Interface | 30001 |
| **Configuration** | |
| TCP & UDP port | 32640 |
| Directory Service | pbrpc://localhost:32638 |
| Debug Level | 6 |
| Statistics | |
| **Load** | |
| # client connections | 0 |
| # pending client requests | 0 |
| Preproc Stage queue length | 0 |
| Storage Stage queue length | 0 |
| Deletion Stage queue length | 0 |
| Open files | 0 |
| **Transfer** | |
| # object written | 0 |
| # object read | 0 |
| bytes sent | 0 bytes |
| bytes received | 0 bytes |
| # files deleted | 0 |
| # replicated object written | 0 |
| bytes replicated | 0 bytes |
| **VM Info / Memory** | |
| Free Disk Space | 8.08 GB |
| Memory free/max/total | 90.30 MB / 1.69 GB / 116.81 MB |
| Buffer Pool stats | 8192: poolSize = 4 numRequests = 29 creates = 4 deletes = 0<br>65536: poolSize = 0 numRequests = 0 creates = 0 deletes = 0<br>131072: poolSize = 0 numRequests = 0 creates = 0 deletes = 0<br>524288: poolSize = 0 numRequests = 0 creates = 0 deletes = 0<br>2097152: poolSize = 0 numRequests = 0 creates = 0 deletes = 0<br>unpooled (> 2097152) numRequests = creates = 0 deletes = 0 |
| **Time** | |
| global XtreeemFS time | Fri Jul 29 14:52:28 CEST 2011 (1311943948041) |
| resync interval for global time | 60000 ms |
| local system time | Fri Jul 29 14:52:28 CEST 2011 (1311943948033) |
| local time update interval | 50 ms |
| **UUID Mapping Cache** | |
| test-localhost-OSD -> pbrpc://localhost/127.0.0.1:32640 pbrpcu://localhost/127.0.0.1:32640 - STICKY | |
| **Detailed Status** | |

List of active replicated files
Full stack trace (all threads)

Figure 3.1: OSD status web page

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.

### 3.3.3 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`.

## 3.4 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. 3.2.5).

If an error occurs, please check if all of the following requirements are 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. 5.2.1).
- A network interface is available on the host. It may be either bound to an IPv4 or IPv6 address.

# Chapter 4

# 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.

## 4.1 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.

### 4.1.1 Prerequisites

To install XtreemFS on Linux, please make sure that FUSE 2.6 or newer, boost 1.35 or newer, openSSL 0.9.8 or newer, libattr 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.

### 4.1.2 Installing from Pre-Packaged Releases

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

```
$> rpm -i xtreemfs-client-1.3.x.rpm
```

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

```
$> dpkg -i xtreemfs-client-1.3.x.deb
```

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

### 4.1.3 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:

- RPM-based distros:

  ```
  cmake >= 2.6
  gcc-c++ >= 4.1
  fuse >= 2.6
  fuse-devel >= 2.6
  boost-devel >= 1.35
  openssl-devel >= 0.9.8
  libattr-devel >= 2
  ```

- DEB-based distros:

  ```
  cmake (>= 2.6)
  build-essential (>=11)
  libfuse-dev (>= 2.6)
  libssl-dev (>= 0.9)
  libattr-dev (>= 2)
  libboost-system1.35-dev or later
  libboost-thread1.35-dev or later
  libboost-program-options1.35-dev or later
  libboost-regex1.35-dev or later
  ```

When done, execute

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

to complete the installation of XtreemFS.

## 4.2 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.

### 4.2.1 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 authorization policy (see Sec. 7.2). If not specified, POSIX permissions/ACLs will be chosen by default. Unlike most other policies, authorization policies cannot be changed afterwards.

In addition, it is recommended to set a default striping policy (see Sec. 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 `--admin_password <password>` to the `mkfs.xtreemfs` command.

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

### 4.2.2 Deleting Volumes

Volumes can be deleted with the `rmfs.xtreemfs` tool. Deleting a volume implies that *any data, i.e. all files and directories on the volume are irrecoverably lost*! Please see `man rmfs.xtreemfs` for a full list of options and usage. Please also note that `rmfs.xtreemfs` does not dispose of file contents on the OSD. To reclaim storage space occupied by the volume, it is therefore necessary to either remove all files from the volume before deleting it, or to run the cleanup tool (see Section 5.2.2).

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.

### 4.2.3 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
```

## 4.3 Accessing Volumes

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

### 4.3.1 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 `mount.xtreemfs` 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@localhost:32638/xtreemfs on /xtreemfs type fuse (...)
```

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

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

### 4.3.2 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.

## 4.4 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. 3.2.5).

**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>`.

# Chapter 5

# XtreemFS Tools

To make use of most of the advanced XtreemFS features, XtreemFS offers a variety of 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.

## 5.1 Installation

The user tools are built, packaged and installed together with the XtreemFS client. For details on how to install the XtreemFS client, please refer to Section 4.1.

To install XtreemFS admin tools, 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 admin tools are only available for Linux.

### 5.1.1 Prerequisites

For the pre-packaged release, you will need Sun Java JRE 1.6.0 or newer to be installed on the system. Some tools also require the `attr/libattr` package to be installed.

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.

### 5.1.2 Installing from Pre-Packaged Releases

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

```
$> rpm -i xtreemfs-tools-1.3.x.rpm xtreemfs-backend-1.3.x.rpm
```

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

```
$> dpkg -i xtreemfs-tools-1.3.x.deb xtreemfs-backend-1.3.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-1.3.x.rpm \
   xtreemfs-backend-1.3.x.rpm
```

on RPM-based distributions,

```
$> dpkg -i --ignore-depends java6-runtime \
   xtreemfs-tools-1.3.x.deb xtreemfs-backend-1.3.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`.

### 5.1.3 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.

## 5.2 Admin Tools

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

### 5.2.1 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 pbrpc://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 pbrpc://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 `--admin_password`; for further details, check the `xtfs_mrcdbtool` man page.

### 5.2.2 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 pbrpc://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 pbrpc://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`.

### 5.2.3 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 pbrpc://localhost:32638 \
   u2i3-28isu2-iwuv29-isjd83 online
```

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

### 5.2.4 Snapshots

XtreemFS is capable of taking file system snapshots. A snapshot caputures an instantaneous image of all files and directories in a volume, which can later be accessed in a read-only manner.

Snapshots can be created, listed and deleted with the `xtfs_snap` tool. A mounted volume is necessary to run the tool; information on how to mount volumes can be found in Section 4.3.

As snapshots cause an additional storage and I/O overhead since they require copy-on-write versioning of files across the OSDs, it is first necessary to enable them on a volume. Snapshots can be enabled as follows:

```
$> xtfs_snap --enable -d /path/to/mounted/volume
```

Once snapshots have been enabled, a snapshot named `mySnapshot` can be taken as follows:

```
$> xtfs_snap -c -r -d /path/to/mounted/volume/subdirectory \
   mySnapshot
```

The optional `-r` parameter enables a recursive capturing that includes all subdirectories beneath the XtreemFS directory `subdirectory`.

A list of all snapshots that exist on the volume can be displayed as follows:

```
$> xtfs_snap -l -d /path/to/mounted/volume
mySnapshot
```

Snapshots are exposed as read-only volumes. To access a snapshot, it is necessary to mount it. The volume name is composed of the original volume name and the snapshot name, separated by an `@` character. Mounting a snapshot works as follows:

```
$> mount.xtreemfs localhost/volume@mySnapshot \
   /path/to/mounted/volume2
```

A mounted volume snapshot can be browsed normally, and all files can be read as on the original volume. However, any attempt to write data on a snapshot will result in an `EPERM` error.

A snapshot `mySnapshot` that is no longer needed can be removed as follows:

```
$> xtfs_snap -x -d /path/to/mounted/volume mySnapshot
```

Please be aware that removing a snapshot does not automatically reclaim storage space from all prior versions. To dispose of obsolete and redundant versions on a specific OSD, it is necessary to perform a version cleanup run with the `xtfs_cleanup` tool:

```
$> xtfs_cleanup -dir localhost:32638 -v \
   uuid:8bca70da-c963-43c7-b30b-d0d605d39fa7
```

**Note:** A snapshot only captures a file in its current state if it is *closed*. Files that are *open* when taking a snapshot are captured in the last state in which they were before they were opened. Since files are implicitly closed on an OSD through a timeout rather than an explicit `close` call, it may happen that files are not included in a snapshot despite having been closed at application level before the snapshot was taken. To make sure a change to a specific file is included in a subsequent snapshot, it is necessary to wait for the close timeout on the OSD before taking the snapshot, which by default is set to 60 seconds.

## 5.3 User Tools

Since release 1.3, all user tools have been replaced by the `xtfsutil` tool. `xtfsutil` displays XtreemFS specific file and directory information, manages file replicas and volume policies.

### 5.3.1 `xtfsutil` for Files

When called without any option `xtfsutil` prints the XtreemFS specific information for a volume, directory, softlink or file.

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

will produce output similar to the following:

```
Path (on volume)     /test.txt
XtreemFS file Id     1089e4fb-9eb9-46ea-8acf-91d10c2170e3:2
XtreemFS URL         pbrpc://localhost:32638/xtreemfs/test.txt
Owner                bjko
Group                users
Type                 file
Replication policy   WqRq
XLoc version         0
Replicas:
  Replica 1
     Striping policy     STRIPING_POLICY_RAID0 / 1 / 128kB
     Replication Flags   partial
     OSD 1               test-osd1/130.73.78.138:32641
  Replica 2
     Striping policy     STRIPING_POLICY_RAID0 / 1 / 128kB
     Replication Flags   partial
     OSD 1               test-osd0/130.73.78.138:32640
  Replica 3
     Striping policy     STRIPING_POLICY_RAID0 / 1 / 128kB
     Replication Flags   partial
     OSD 1               test-osd2/130.73.78.138:32642
```

The fileID is the unique identifier within XtreemFS, e.g. used by the OSD 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). The XtreemFS URL shows you on which MRC the volume is hosted and the name of the volume. This file has three replicas and is replicated with the `WqRq` policy (majority voting).

### Changing the Replication Policy

The replication policy defines how a file is replicated. The policy can only be changed for a file that has no replicas. If you wish to change the policy for a replicated files, you have to remove all replicas first.

To change the replication policy, execute `xtfsutil` with the following options:

```
$> xtfsutil --set-replication-policy ronly /xtreemfs/test.txt
```

The following values can be passed to `--set-replication-policy`:

**none** File is not replicated.

**ronly** File is read-only replicated, the file cannot be modified.

**WqRq** File is read-write replicated and can be modified. Updates are sent to a majority, replicas can fail.

**WaRa** Like WqRq but updates are sent to all replicas and no replica may fail.

### Adding and Removing Replicas

Replicas can be added for files that have a replication policy defined, i.e. not `none`. When adding a replica, you need to specify on which OSD to create the new replica. Alternatively, you can use `auto` instead of an OSD UUID. With `auto` set, the `xtfsutil` will automatically select an OSD.

To add a replica execute:

```
$> xtfsutil --add-replica auto /xtreemfs/test.txt
```

For read-only replicated files, replicas are partial by default. To create a full replica, you can use the `--full` flag when adding a replica. For read-write replicated files, all replicas are equal and there is no further options.

In case you want to select an OSD for a new replica manually, you can retrieve a list of up to 10 OSDs for a file. The MRC automatically filters and sorts the list of OSDs depending on the policies set for a volume. In addition, the MRC also excludes all OSDs that already have a replica of that file. To retrieve this list execute:

```
$> xtfsutil --list-osds /xtreemfs/test.txt
OSDs suitable for new replicas:
  test-osd1
  test-osd2
```

To remove a replica, pass the OSD's UUID to `xtfsutil`:

```
$> xtfsutil --delete-replica test-osd1 /xtreemfs/test.txt
```

### 5.3.2 `xtfsutil` for Volumes

To display the volume policies and settings, execute `xtfsutil` on the mountpoint without any options.

```
$> xtfsutil /xtreemfs
```

will produce output similar to the following:

```
Path (on volume)     /
XtreemFS file Id     1089e4fb-9eb9-46ea-8acf-91d10c2170e3:1
XtreemFS URL         pbrpc://localhost.localdomain:32638/replicated
Owner                bjko
Group                users
Type                 volume
Free/Used Space      24 GB / 6 bytes
Num. Files/Dirs      1 / 1
Access Control p.    2
OSD Selection p.     1000,3002
Replica Selection p. default
Default Striping p.  STRIPING_POLICY_RAID0 / 1 / 128kB
Default Repl. p.     WqRq with 3 replicas
```

#### Changing the Default 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.

The striping policy can be changed with `xtfsutil` as follows:

```
$> xtfsutil --set-dsp -p RAID0 -w 4 -s 256 /xtreemfs
```

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

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.

### Changing the Default Replication Policy

The Default Replication Policy defines how new files on a volume are replicated. This policy can be set on the volume and is valid for all sub-directories. It affects only new files and doesn't modify the replication settings for existing files.

The replication policy can be changed as follows. In this example, all files will have three replicas with WqRq mode.

```
$> xtfsutil --set-drp --replication-policy WqRq \
            --replication-factor 3 /xtreemfs
```

The following values can be passed to `--replication-policy`:

**none** New files are not replicated.

**ronly** Files are initially created without replicas and can be modified until they are closed. On close, the file is set to read-only and the replicas are created. Replicas are partial by default, full replicas will be created if the `--full` flag is set.

**WqRq, WaRa** New files are read-write replicated and can be modified.

## 5.3.3 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. 7.3.

The policies are set and modified with the `xtfsutil` tool on the volume (mount point). When called without any options, `xtfsutil` will also show the policies currently set for the volume. A policy that controls the selection of a replica is set as follows:

```
$> xtfsutil --set-rsp dcmap /xtreemfs
```

This will change the current replica selection policy to a policy based on a data center map.

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:

```
$> xtfsutil --set-osp 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. 7.3 and `man xtfsutil` for details):

`default` The default OSD selection policy selects a random subset of OSDs that are responsive and have more than 2GB of free disk space.

`fqdn` Selects OSDs based on the size of the postfix match of the fully qualified domain names and on the free space.

`dcmap` Selects OSDs based on the distance defined in the datacenter map and on the free space.

`vivaldi` Selects OSDs based on the distance of the vivalid coordinates between client and OSD and on the free space.

In addition, custom policies can be set by passing a list of basic policy IDs to be successively applied instead of a predefined policy name.

### 5.3.4 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 Section 7.3. Policy attributes can be set as follows:

```
$> xtfsutil --set-pattr domains --value "*.xtreemfs.org bla.com" \
   /xtreemfs
```

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

```
$> xtfsutil --list-pattrs /xtreemfs
```

### 5.3.5 Modifying Access Control Lists

In some cases, it may be necessary to enforce access control on a file or directory at a finer granularity than expressible with simple "rwx"-like access rights. XtreemFS supports Access Control Lists (ACLs) to set individual access rights for users and groups.

An ACL entry for the user `someone` with the value `rx` ("read or execute") can be added as follows:

```
$> xtfsutil --set-acl u:someone:rx /xtreemfs
```

An existing entry can be removed as follows:

```
$> xtfsutil --del-acl u:someone /xtreemfs
```

Please be aware that when files or directories are accessed, the actual evaluation of ACL entries depends upon the effective authorization policy on the volume (see Section 7.2). With a POSIX authorization policy, ACL entries will be evaluated as described at http://www.suse.de/~agruen/acl/linux-acls/online.

## 5.4 Vivaldi

**Attention:** Vivaldi is currently not included in Release 1.3.0 but will become available in one of the next minor releases. 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_coordinates \
   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. 5.3.3 for details.

## 5.5 Test Tools

XtreemFS provides two tools to simplify testing. `xstartserv` can be used to start and stop XtreemFS servers manually. `xtestenv` automatically sets-up an entire test environment with servers and mounted clients. In addition, `xtestenv` can be used to execute the automatic integration tests.

# Chapter 6

# Replication

XtreemFS offers replication of all data. On the one hand, the Directory Service (DIR) and the Metadata Catalog (MRC) are replicated via BabuDB database replication. On the other hand, files are replicated on the OSDs with read/write or with read-only replication. In this chapter, we describe how these replication mechanisms work, their requirements and potential use-cases.

## 6.1 Read/Write File Replication

File that are replicated with read/write replication have the same semantics as non-replicated files. That means that all operations can be executed on those files and that data is kept consistent across replicas. Applications and users won't see a difference between read/write replicated and regular files.

Internally, the read/write replication is implemented using the primary/backup approach with leases. When a file is opened, all OSDs that have a replica "talk" to each other to decide which replica becomes the *primary*. In XtreemFS we use leases for the primary election, this means that an OSD will become primary for some time. If it fails, the lease times out and another OSD can become primary. Once a replica has acquired the lease to become primary, it checks with the other replicas to ensure all replicas are in a consistent state. After this so called replica reset phase, the primary processes client operations. Reads can be executed locally on the primary. However, operations that modify data such as write and truncate, are executed on the primary which passes these updates on to the other replicas (backups).

The replication of files adds significant communication overhead to keep replicas in sync. When a file is opened, the OSD which the client contacts requires at least three message round-trips to acquires the lease and to execute the replica reset. Once a primary was elected, read operations can be executed locally without any communication. Truncate and write require a single round-trip between the primary and the backup OSDs.

Depending on the selected replication policy, the read/write replication can tolerate some replica failures. The **WqRq** policy employs majority voting and can tolerate replica failures as long as a majority of replicas is available. This is the most fault-tolerant strategy in XtreemFS. However, guarantees only that data is stored on a

majority of the replicas. If you lose more replica permanently, data might be lost. The **WaRa** policy writes updates to all replicas which yields higher data safety. However, this policy cannot tolerate replica failures.

Due to the communication overhead, the read-write replication should only be used for up to ten replicas. If you need more replicas or if you need replicas for caching, you should consider the read-only replication.

## 6.2 Read-Only File Replication

The read-only is designed for use-cases where you have many replicas that are not modified. Since files cannot be changed, the replicas don't need to be coordinated. Therefore, this replication mode can handle as many replicas as you like, e.g. to create copies of files close to consumers. One use-case for the read-only replication is to build a content-distribution network (CDN) like infrastructure.

Read-only replicas are either *full* or *partial*. Full replicas immediately copy the file data from other replicas when they are created. XtreemFS uses a rares-first strategy (similar to BitTorrent) to increase the replication factor as quickly as possible. In contrast, partial replicas are initially empty and fetch the file data (objects) on demand when requested by a client. Partial replicas also pre-fetch a small number of objects to reduce latency for further client reads.

Files that are read-only replicated can only be opened in read-only mode and cannot be modified. To allow existing applications to take advantage of the read-only replication without modifications, XtreemFS offers "replicate-on-close". When the default replication policy for a volume is set to "ronly", files can be opened and modified like regular files until they are closed. Once a file is closed, it is set to read-only and is replicated according to the replication factor set for the volume. This mode should, however, not be used for data safety as there are no guarantees that all replicas were created successfully when the `close()` operation returns. For data safety, please use read-write replication.

## 6.3 BabuDB Database Replication (DIR/MRC)

FIXME!!!

# Chapter 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.

## 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:

### 7.1.1 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`.

### 7.1.2 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`).

## 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.

## 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. 5.3.3.

### 7.3.1 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.

### 7.3.2 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.

## 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. **??**.

## 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. 3.2.3, 3.2.5), 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.

# Appendix A

# Support

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

# Appendix B

# Hadoop Integration

## B.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.

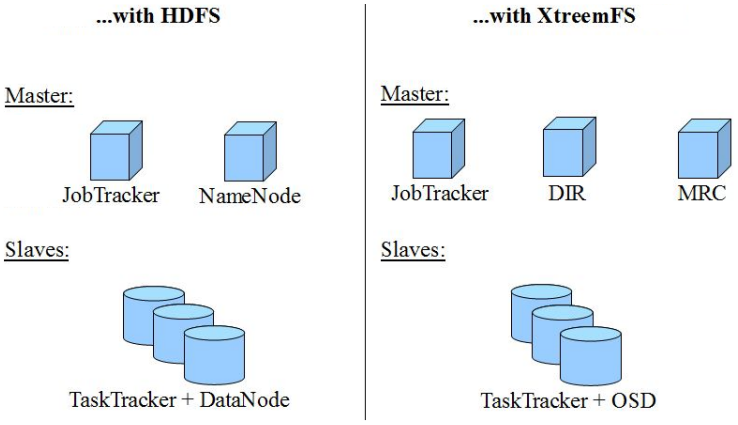



Figure B.1: Hadoop cluster setup recommendation

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.

## B.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) inlcuding 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. 1. 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.</description>
</property>

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

# Appendix C

# Command Line Utilities

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

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

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

**rmfs.xtreemfs (formerly xtfs_rmvol)** Deletes a volume.

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

**xstartserv** Tool for manually starting/stopping XtreemFS servers, e.g. for testing and development.

**xtestenv** Tool for automatic set-up of a test environment and for executing the autotests.

**xtfsutil** XtreemFS's swiss army knife.

**xtfs_cleanup** Deletes orphaned objects on an OSD, restores orphaned files and removes obsolete file versions.

**xtfs_snap** Creates, lists and deletes snapshots.

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

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

**xtfs_vivaldi** client service to calculate vivaldi coordinates.

# Index


Access Policy, 54
    Authorize All, 54
    POSIX ACLs, 54
    POSIX Permissions, 54
    Volume ACLs, 54
allow_others option, 36
allow_root option, 36
Architecture, 4
Authentication, 4
Authentication Provider, 9, 53
    NullAuthProvider, 53
    SimpleX509AuthProvider, 54
Authorization, 4
Authorize All Access Policy, 54

CA
    Certificate Authority, 11
Certificate, 4, 10
Certificate Authority, 11
Client, 5
Create Volume, 35
Credentials, 10

Delete Volume, 35
DIR, 4
Directory Service, 4

fileID, 45
FUSE, 5

Hadoop
    Integration, 61

init.d, 29

Java KeyStore, 11
JKS, 11

Metadata, 4
Metadata and Replica Catalog, 4
Metadata Server, 4
mkfs.xtreemfs, 35
Mount, 36
mount.xtreemfs, 36
Mounting, 5
MRC, 4

NullAuthProvider, 53

Object, 4
Object Storage Device, 4
Object-based File System, 4
OSD, 4
OSD Selection Policy, 55

PKCS#12, 10
Policy
    Access Policy, 54
    OSD Selection Policy, 55
    Striping Policy, 4, 57
POSIX ACLs Access Policy, 54
POSIX Permissions Access Policy, 54

RAID0, 3, 57
rmfs.xtreemfs, 35

SimpleX509AuthProvider, 54
SSL, 4
Status Page, 29
Storage Server, 4
Stripe Size, 58
Striping, 57
    Stripe Size, 58
Striping Policy, 4, 57
Striping Width, 57

umount.xtreemfs, 36
Unmount, 36
user_allow_other option, 36
UUID, 9

VFS, 5
Volume, 4, 5
    Create, 35

Delete, 35
Mount, 36
Un-mount, 36
Volume ACLs Access Policy, 54

X.509, 4, 10
xtfs_mkvol, 35
xtfs_mount, 36
xtfs_rmvol, 35
xtfs_umount, 36