XRootD

Proxy Storage Services (Caching & Non-Caching) Configuration Reference

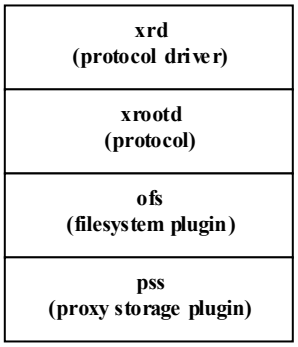
11 January 2017

Andrew Hanushevsky (SLAC)
Alja Mrak-Tadel (UCSD)

1 Introduction

This document describes **XRootD** configuration directives for the **P**roxy **S**torage **S**ervice (**pss**) components.

Configuration directives for each component come from a configuration file. The **xrootd** structure requires that all components read their directives from the same configuration file. This is the configuration file specified when **xrootd** was started (see the **xrootd –c** command line option). This is possible because each component is identified by a unique 3-letter prefix. This allows a common configuration file to be used for the whole system.



The particular components that need to be configured are the file system plug-in (**ofs**) and the proxy storage plug-in (**pss**). The relationship between **xrootd** and the plug-ins is shown on the left. The protocol driver (**xrd**) runs the **xroot** protocol which, in turn, utilizes the file system plug-in that, itself, relies on the proxy system plug-in. Collectively, this is called **xrootd**, the executable that encapsulates all of the components.

The prefixes documented in this manual are listed in the following table. The **all** prefix is used in instances where a directive applies to more than one component. *Records that do not start with a recognized identifier are ignored*. This includes blank record and comment lines (i.e., lines starting with a pound sign, #).

Prefix	Component
ofs	**O**pen **F**ile **S**ystem coordinating **acc**, **cms**, **oss** & **pss** components
pfc	Proxy File Cache (i.e. specialized proxy plug-in)
pss	**P**roxy **S**ystem **S**ervice (i.e., specialized **oss** plug-in)
all	Applies the directive to the above components.

Refer to the manual "**Configuration File Syntax**" on how to specify and use conditional directives and set variables. These features are indispensable for complex configuration files usually encountered in large installations.

By default, all special features are disabled. A progression of directives enables certain features. Features are enabled as described in the following table.

Directive	Purpose
`ofs.osslib`	Load the shared library implementing a special **pss** (storage system) component.

When you are not running a clustered set of proxy servers, you need to specify the **ofs.osslib** directive. The plug-in is encapsulated in **libXrdPss.so** and is installed in the standard location. If you are clustering proxy servers (i.e. you specified the all.role directive indicating you are clustering proxies, **libXrdPss.so** is automatically loaded as the **ofs.osslib** plug-in.

To properly configure an xrootd proxy you need to review **xrd**, **xrootd**, and **ofs** directives and specify the ones that are relevant to your installation. The **xrd** and **xrootd** directives can be found in the "Xrd/XRootd Configuration Reference" while the ofs directives can be found in the "Open File System & Open Storage System Configuration Reference". If you are configuring a proxy cluster then you will need to specify certain cms directives. These can be found in the "Cluster Management Service Configuration Reference".

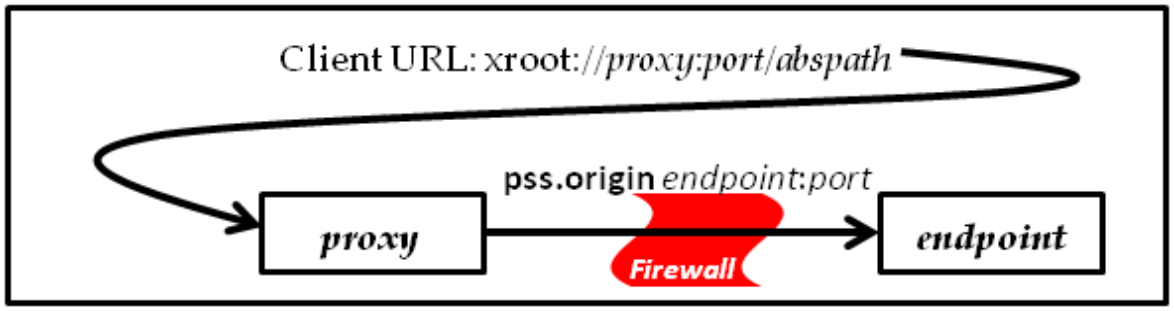
The following section describes the standard proxy service. This proxy service is primarily intended to be used for LAN access to bridge firewalls for remote clients. While it includes a memory caching component that can improve performance for WAN access in limited scenarios, it is not intended to be used as a true WAN proxy. In a subsequent section the disk caching proxy is described. This proxy variant is intended to improve WAN access as it is able to cache files or parts of files on disk at a remote location. Caching files or file segments may minimize WAN traffic depending on the specific workload.

Hence, the standard proxy service is intended to be used as a local proxy while the disk caching proxy is intended to be used as a remote proxy.

A proxy service, whether or not it is a disk caching proxy, can be configured in direct mode or forwarding mode. In direct mode the client is limited to initially contacting a particular endpoint. In forwarding mode, the client is free to specify the actual endpoint that is to be used for the initial contact.

1.1 Direct Mode Proxies

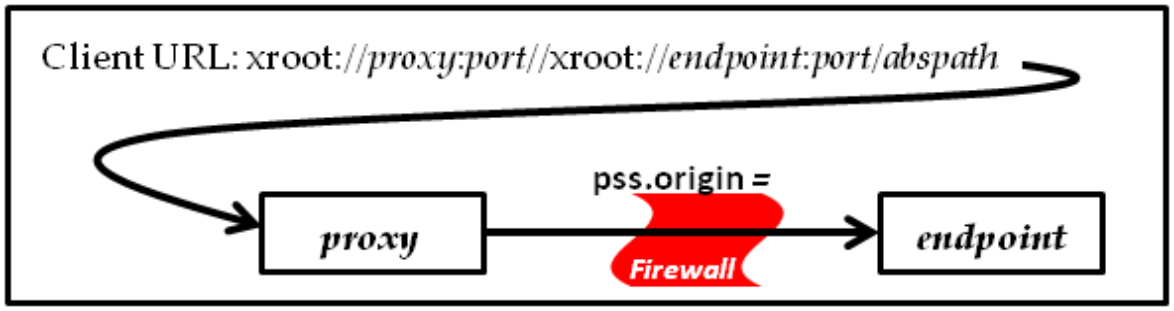
Direct mode proxies are the most common kind of proxies and are suitable to directly fronting an **XRootD** cluster. The **pss.origin** directive specifies the initial point of contact. The following diagram illustrates a direct mode proxy.



In the diagram the client present a URL that identifies the proxy server or proxy cluster. The proxy then contacts the data origin, a particular endpoint, on behalf of the client and performs the requested operation. The endpoint can be an actual server or a cluster redirector. This mode is suitable for incoming connections to a fire walled cluster or outgoing connections to a particular data source.

1.2 Forwarding Mode Proxies

Forwarding mode proxies are specialized proxies that allow the client to specify the actual endpoint to be used for a request and suitable in cases where the client determines the actual endpoint that must be used. The **pss.origin** directive specifies that the endpoint will be supplied by the client, as shown below.



In the diagram the client prefixes the destination URL with the proxy location. The proxy server then contacts the specified endpoint on behalf of the client. The client is free to specify any valid endpoint. This mode is most suitable for outgoing connections and poses certain security issues that are described in a subsequent section. A valid destination URL must specify the **xroot** or **root** protocol element.

1.2.1 Forwarding Pile Paths

When configuring a forwarding proxy you should realize that from the proxy's perspective the URL suffix supplied by the client is initially taken as an actual path. This means that you

a) must export at least some portion of the URL suffix, and
b) can perform authorization operations on the specified URL (see the Security Reference for more information).

Exports of the form

all.export /xroot:/ *options*
all.export /root:/ *options*

satisfy the export requirement (see the **OFS** reference manual for a description of the **all.export** directive). You can specify longer export names in order to restrict the endpoints that can be contacted using the forwarding proxy. However, since multiple slashes are compressed before comparing the URL against the exported path list, you must not specify successive slashes. For example, say you wanted to restrict forwarding to only the host **foobar:1094**, then the exports would be specified as

all.export /xroot:/foobar:1094/
all.export /root:/foobar:1094/

Even though the client would actually suffix "/xroot://foobar:1094/" to the proxy's URL location. The same applies to entries placed in the authorization file. All double slashes must be removed in order for authorization rules to be matched. In both cases, the client's destination must exactly match the export specification. As a side note, error messages in the proxy's log file will show URL portions of the path without multiple slashes and should not cause any alarm.

A better alternative is to use the **pss.permit** directive to restrict the set of target destinations. This allows client to specify the target host in a variety of syntactic ways since the permit applies to the actual resolved destination IP address. In this case you only need to export the protocol segment of the destination URL.

1.2.2 Forwarding Object IDs

To exporting object IDs (i.e. object names without a leading slash) via a forwarding proxy, the configuration file must contain one of the following two export directives:

all.export *
or **all.export *?**

See the "**Xrd/XRootD Reference**" for more information on using object IDs. Since object IDs do not start with a slash; hence, cannot be confused with a file path, the specified client destination URL must appear like an object ID (i.e. it may not start with a slash). Using an absolute file path URL to forward an object ID converts the object ID to a file path and access will likely fail.

Additionally, you can neither use the authorization framework nor the **all.export** directive to restrict forwarding by destination. Instead you must use the **pss.permit** directive. This is because object IDs are arbitrary and are only recognized as such in only certain restricted contexts.

The following table compares forwarding an absolute path to an Object ID to a server named "foobar:4096" via "myproxy:1094" (notice the missing double slashes for myobject before the second "xroot" and the pathname, myobject).

Access Target	Client URL Specification
/my/data	xroot://myproxy:1094//xroot://foobar:4096//my/data
myobject	xroot://myproxy:1094/xroot://foobar:4096/myobject

A forwarding proxy server is capable of accepting file paths as well as object IDs as long as the correct exports are specified.

1.2.3 Security Considerations

Proxy servers never forward client credentials to the endpoint that they contact. Instead, they use their own credentials to establish authenticity. This works well for incoming connections as the proxy can validate the client before performing any actions on behalf of the client. However, outgoing connections pose a problem; especially for forwarding proxies which are meant to be used for outgoing connections. Validating a client using internal site rules does not necessarily capture any specific access restrictions in effect outside the site. For instance, a site may authorize an outgoing client but that client may not be actually authorized by the final endpoint. If the proxy itself is authorized by the final endpoint then, in effect, the client gets access to resources that would otherwise be prohibited if the client made direct contact with the outside endpoint. Consequently, great care must be given when providing resources using a forwarding proxy.

1.3 Combination Mode Proxies

A proxy server can be configured in direct more as well as forwarding mode. This is known as a combination mode proxy and is done via a special form of the **pss.origin** directive. For instance, specifying

pss.orgin = *host***:***port*

Allows a client to forward a connection via a URL type of path or connect to a particular destination (i.e. *host*:*port*) if the path is not a URL. The exports must allow such combinations. For instance, the trio of

all.export /xroot:/ *options*
all.export /root:/ *options*
all.export /atlas *options*

indicates that **xroot:** and **root:** URLs should forward the request to the client-specified destination while paths starting with "/atlas" connect to the destination specified in the **pss.origin** directive.

In practice, if it is sufficient for the destination host to enforce its exports the whole specification can collapse to a single specification

all.export / *options*

as the proxy recognizes a URL if it begins with "xrootd:/" or "root:/" and treats anything else as a regular path.

1.4 Minimal Sample Configuration Files

```
# Specify that we are a direct mode proxy fronting the host
# data.stanford.edu (server or redirector)
#
pss.origin data.stanford.edu

# The export allows access to any path via proxy as the
# origin host will enforce its own exports
#
all.export /

# We need to load the proxy plugin for this to actually work
#
ofs.osslib libXrdPss.so
```

Minimal Direct Mode Proxy Configuration

```
# Specify that we are a forwarding proxy
#
pss.origin =

# The export allows xroot and root type URL's destinations
#
all.export /xroot:/
all.export /root:/

# We need to load the proxy plugin for this to actually work
#
ofs.osslib libXrdPss.so
```

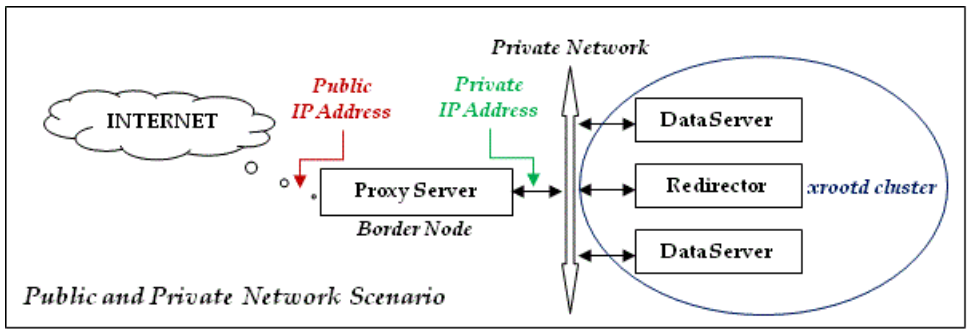
Minimal Forwarding Mode Proxy Configuration

```
# Specify that we are a forwarding proxy for URL type of paths
# and a direct mode proxy for any other type of path
#
pss.origin = data.stanford.edu

# The export allows any path to be specified. However, paths
# starting with "/xroot:/" or "/root:/" are treated as URLs
# and are forwarded. Anything else is accessed via
# data.stanford.edu
#
all.export /

# We need to load the proxy plugin for this to actually work
#
ofs.osslib libXrdPss.so
```

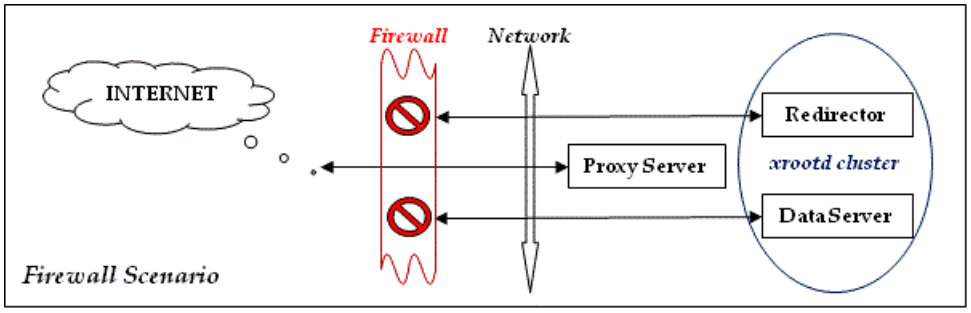
Minimal Combination Mode Proxy Configuration

2 Standard Proxy Service

The **ofs** layer can provide a proxy service by using a special **oss** plug-in, "libXrdPss.so". A proxy service is required if wish to provide public internet access to data served by **xrootd** servers that can only access a private network or sit behind a firewall. You do not need to setup a proxy service if all of your **xrootd** servers are accessible via the public network.



As shown in the diagram to the left, a proxy server must have access to the public as well as private or firewalled network. Typically, machines with this capability are called border nodes.



A border node is a computer that either has access to the public as well as the private network using two distinct IP addresses or one that is allowed to access the public network through a firewall using a single local IP address.

The particular scenario in effect is a local option determined by your particular network setup. The biggest challenge is to properly setup the network configuration; a task best left to networking personnel.

Two IP address scenarios require static routes to be established within the border machine so that the local **xrootd** cluster is always found using the private network while internet traffic uses the public network. Single IP address scenarios (i.e. firewalled networks) require that special firewall rules be established to allow internet access to the proxy server on the **xrootd** port while disallowing internet access to the local **xrootd** cluster.

Once routes or firewall rules have been established, you should use the **netchk** utility, found in the **xrootd utils** directory, to see if you have access from an external public node all the way to the redirector and each data server node. This verifies that network routing or firewall rules have been correctly established. In order to use **netchk**, **perl** must be installed on each machine and you must be able to do an **ssh** login to each one of them. The utility is described in the last subsection of this section.

The following sections describe how to setup a simple proxy server as well as a proxy server cluster. Additionally, various tuning options are also described. These options are independent of whether you setup a simple proxy or a proxy cluster.

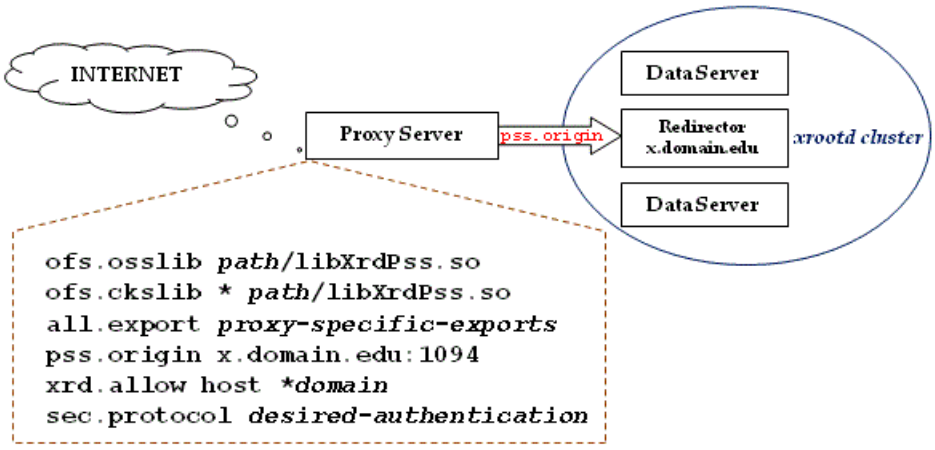
2.1 Simple Proxy Server

The simple proxy server is configured as a non-clustered **xrootd** server. This means you neither specify the **all.role** nor the **all.manager** directives documented in the "Cluster Management Service Configuration Reference". Other directives may be specified at your discretion.

You control which paths are publically available using the **all.export** directive that is specific to the proxy server (i.e. the proxy will only allow access to the specified paths). This also means that data servers can provide read-write access to particular paths while the proxy server can only provide read-only access to some or all such paths. The same holds true for the **stage** attribute. You can prohibit staging files via the proxy by exporting the files via the proxy with the **nostage** attribute.

The **xrd.allow** directive can be used to limit which range of hosts can use the proxy server; while the **sec.protocol** directive can be used to limit access to the proxy server to clients that can only be properly authenticated. These directives do not differ from those you would use in a standard **xrootd** cluster and documentation can be found the **xrootd** and security reference manuals.

Very few directives are needed to setup a simple proxy server. A sample configuration file matching the illustrated cluster setup is shown below. Directives shown in **red** are ***required***; those shown in **green** are ***optional*** and serve to limit who can actually use the proxy server.



The **ofs.osslib** directive tells the **ofs** layer to use the plug-in that changes a regular **xrootd** server into a proxy server. Normally, the plug-in is named "libXrdPss.so". You must specify the location of this shared library plug-in. To support checksum calculations you must also indicate that the checksum manager is a proxy as well. This is done loading the proxy library via the **ofs.ckslib** directive.

The **all.export** directive restricts the proxy server to those paths you want to be publically accessible. Most likely, you will also designate these paths as read-only and nostage regardless of how they are locally served within the cluster.

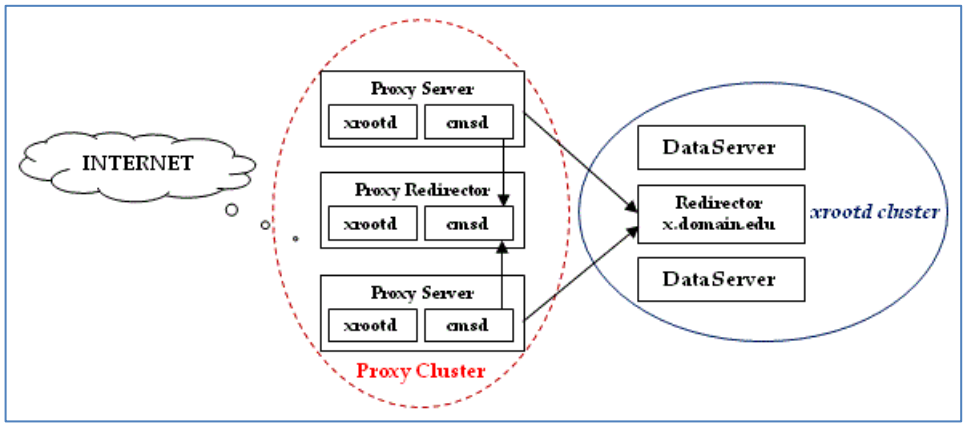
The **pss.origin** directive tells the plug-in where the xrootd cluster's redirector is located (in the example it is x.domain.edu). The port number must match the port number used by your local clients (in the example this is 1094). Indeed, the proxy server is, in fact, just another local client that is capable of serving data across the public internet.

The two green directives, **xrd.allow** and **sec.protocol**, are used to limit who can use the proxy server. The **xrd.allow** directive can restrict access by domain when you use the asterisk notation. The **sec.protocol** directive can restrict access to authenticated clients. While neither are required, your particular security requirements may force you to specify one or both of them.

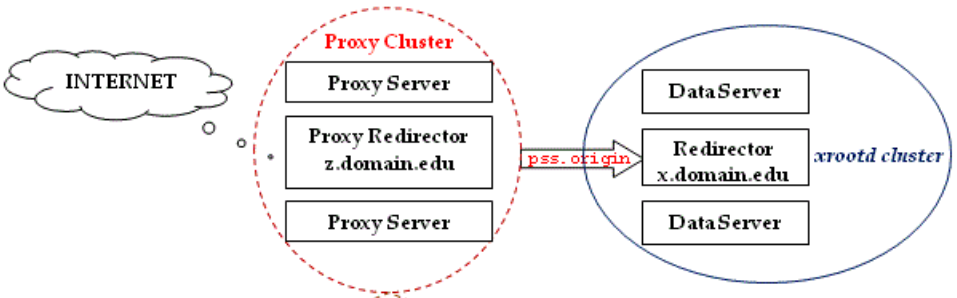
2.2 Proxy Cluster

A proxy cluster consists of two or more simple proxy servers arranged as an **xrootd** cluster. While one could use a load-balancing DNS to bridge together multiple proxy servers, using the **xrootd** cluster management infrastructure to do this provides not only true load balancing but also fail-over. That is, should a proxy server become unresponsive an **xrootd** client will automatically switch to another working proxy server. This also allows you to take down proxy servers for maintenance without worrying if they are currently in use, as long as one proxy server remains.

The mechanics of setting up a proxy cluster is largely the same as setting up a regular **xrootd** cluster. However, different parameters are specified for the **all.role** and **all.manager** directives. Additionally, unlike a simple proxy server you must also run a **cmsd** daemon everywhere you run an **xrootd** proxy server and you must configure a proxy redirector. The initial point of contact for external clients is the proxy redirector. This is shown in the adjacent diagram.



Setting up a proxy cluster is a minor variation of setting up a standard **xrootd** cluster. For proxy servers you need to identify the proxy manager and tell each proxy server that it is part of a proxy cluster. This allows the proxy servers to cluster around the proxy manager and the proxy manager then knows how to redirect clients to a working proxy server. A sample configuration file follows. You should compare it to one for a simple **xrootd** data cluster.




```
all.exort proxy-specific-exports
pss.origin x.domain.edu:1094
all.manager proxy z.domain.edu:1213
if z.domain.edu
all.role proxy manager
else
all.role proxy server
fi
if z.domain.edu && exec cmsd
xrd.allow host *.domain.edu
fi

if exec xrootd
xrd.allow host *domain
sec.protocol desired-authentication
fi
```

All **red** directives are required. The **all.export** and **pss.origin** directives are the same as for simple proxy servers. These tell the **xrootd**'s what to expor, and where the **xrootd** data cluster resides, respectively.

The **all.manager** directive tells each proxy server who the proxy redirector is and what port the **cmsd** is using (here we arbitrarily chose 1213).

In the **if-fi** construct roles are assigned to each server. Since z.domain.edu is the redirector, its **role** is **proxy manager**. All other nodes in the proxy cluster have a **role** of **proxy server**. This also causes the ofs layer to automatically load **libXrdPss.so**, by default, as the **ckslib** and the **osslib**; making the **ofs.ckslib** and **ofs.osslib** directives unnecessary. If the defaults are inappropriate, specify these directives to override the default.

Since we only want true proxy servers to be allowed to connect to the **cmsd** running on the proxy redirector node, the second **if-fi** construct specifies that only those nodes are allowed to connect. This is especially important because the proxy redirector is publically accessible and cmsd connections should be restricted to a known set of hosts. The particular example uses a generic specification. You may wish to make "allow" more specific. For instance, if proxy servers have a DNS name of "proxy*nn*.domain.edu", where *nn* is a sequence number, then it would be better to specify "**xrd.allow host proxy*.domain.edu**".

The green directives, **xrd.allow** and **sec.protocol**, are used to limit who can use the proxy servers. The **xrd.allow** directive can restrict access by domain when you use the asterisk notation. The **sec.protocol** directive can restrict access to authenticated clients. While neither is required, your particular security requirements may force you to specify one or both of them. You should especially note that these directives *apply only* to the **xrootd** and not the **cmsd**. This is why they have been qualified by an **if-fi** clause that applies them only to the **xrootd** proxy servers.

The default load distribution is to round-robin requests across all available proxy servers. You can use the **cms.sched** directive for the proxy redirector along with the **cms.perf** directive for proxy servers to do load balancing on actual load. These directives are described in the **cmsd** reference manual.

When configuring a proxy cluster, all proxy servers must either be in direct mode, forwarding mode, or combination mode. Any attempt to mix modes will lead to unpredictable result.

If you are using disk-caching proxies, a different type of cluster configuration must be used. Refer to the description of disk caching proxy clusters for details.

2.3 Proxy Server Directives

The proxy server plug-in is based on **xrootdFS** which, is based on the **xrootd POSIX** library which, in turn, uses the **xrootd client** library. Each of these components has specific options that can be specified in the same configuration file used for the proxy setup. Specifically, the following options are processed by the proxy

- **all.export**
- **oss.defaults**

Proxy specific directives and options are described below.

```
pss.config workers number

pss.inetmode {v4 | v6}

pss.localroot path

pss.namelib path

pss.origin  dest | = [dest]     required directive

pss.permit [/] [*] hspec

pss.setopt ConnectTimeout seconds
pss.setopt DataServerConn_ttl seconds
pss.setopt DebugLevel {0 | 1 | 2 | 3}
pss.setopt ParStreamsPerPhyConn number
pss.setopt ReconnectWait seconds
pss.setopt RedirectLimit number
pss.setopt RedirectorConn_ttl seconds
pss.setopt RequestTimeout seconds
pss.setopt TransactionTimeout seconds
pss.setopt WorkerThreads number
```

```
dest:     host[:port | port]
```

Where:

namelib

specifies the name- to-name mapping plug-in library. This directive works just like the **oss.namelib** directive but only applies to the proxy server.

origin specifies the location of the redirector or server for which the server being configured is to proxy. ***This directive is required***.

host The DNS name or IP address of the redirector or server being proxied.

port The TCP port number or service name of the redirector or server associated with *host*. The port may be specified with an adjacent colon or space separation. The default is 1094.

= Configures the proxy in forwarding mode where the client supplies the actual endpoint. If a *host* and optional *port* follows the equals sign then this endpoint is used when the client has not specified a **root** or **xroot** URL as an endpoint. Otherwise, the client receives a "not supported" error.

permit

specifies an allowed destination that a client can use via a forwarding mode proxy. If no permit directives are specified, no restrictions apply. Permits, by default, apply to path and objectid URLs.

/ The permit applies only to path-type URLS.

***** The permit applies only to objectid-type URLS.

hspec The DNS host name allowed as a target connection. Substitute for *name* a host name or IP address. A host name may contain a single asterisk anywhere in the name. This lets you allow a range of hosts should the names follow a regular pattern or if you wish to allow connections to all hosts in a particular domain. IP addresses may be specified in IPV4 format (i.e. "a.b.c.d") or in IPV6 format (i.e. "[x:x:x:x:x:x]").

config

Configures various parameters affecting the fuse interface. The defaults are:

workers 16

Otherwise, the following option may be specified:

workers The *number* of parallel threads to be used when collecting information from data servers or issuing requests to multiple data servers due to a single request. The default is 16.

inetmode

Specifies the network stack to use when connecting to other servers. The default is determined by the **-I** command line option; which defaults to **v6**. Options are:

v4 Use the IPV4 network stack to connect to servers. This means only servers reachable by IPV4 addresses may be used.

v6 Use the IPV6 network stack to connect to servers. This means only servers reachable by IPV4 mapped addresses or IPV6 addresses may be used. This is the least restrictive option.

localroot

Configures the default name- to-name mapping using the specified path. The action is identical to that of **oss.localroot** but is only used by the proxy to map file names prior to requesting action from the origin.

setopt

Configures the **xrootd** client used by the proxy top communicate with the origin. Each **setopt** directive allows a single configuration argument. Specify one or more of the following, as needed.

ConnectTimeout

The number of *seconds* to wait for a server connection to complete. The default is 120 seconds.

DataServerConn_ttl

The number of *seconds* to keep an idle data server socket open. The default is 1200 (i.e. 20 minutes).

DebugLevel

The level of debugging. **0,** the default, turns off debug output; increasing values produce more detail.

ParStreamsPerPhyConn

The *number* of parallel TCP streams to use for data server I/O. You can specify a number from **0** to **15**. The default is **2**.

ReconnectWait

The number of *seconds* to wait between server reconnects in the presence of a fatal error. The default is 1800.

RedirectLimit

The maximum number of sequential (i.e. without a break) redirects to other servers that may occur before a fatal error is declared. The default is 16.

RedirectorConn_ttl

The number of *seconds* to keep an idle redirector socket open. The default is 3600 (i.e. 60 minutes).

RequestTimeout

The maximum number of seconds a request can be active without a response before an error is declared. The default is 300 seconds.

RequestTimeout

The maximum number of seconds a request can be active without a response before an error is declared. The default is 300 seconds.

TransactionTimeout

The maximum number of *seconds* of wait-time that can be imposed by the server before an error is declared. The default is 28800 (i.e. 8 hours).

WorkerThreads

The maximum *number* of responses that can be processed in parallel. Each response is handled in a separate thread. The default is 64.

Notes

1) The **inetmode** option allows you to create network protocol stack bridges. Fr instance, if the proxy accepts IPV6 addresses and **inetmode v4** is specified; the proxy becomes a bridge between incoming IPV6 addresses and an IPV4 network.
2) The proxy also accepts environmental variables that control the underlying xrootd client. Some of these environmental variables offer more extensive control over the client's behavior then the **setopt** directive allows. When an environmental variable is set, it over-rides the equivalent specification in the configuration file. The variables are described in the documentation for the **xrdcp** command.

3 Disk Caching Proxy

The disk caching proxy is almost identical to the standard proxy except that a disk is substituted for a memory cache. This makes the proxy more suitable for WAN access. The **libXrdFileCache.so** shared library is a proxy server cache plug-in used for caching of data into local files. Two modes of operation are supported.

The first mode simply pre-fetches complete files and stores them on local disk. This implementation is suitable for optimization of access latency, especially when reading is not strictly sequential or when it is known in advance that a significant fraction of a file will be read, potentially several times. Of course, once parts of a file are downloaded, access speed is the same as it would be for local **XRootD** access.

The pre-fetching is initiated by the file open request, unless the file is already available in full. Pre-fetching proceeds sequentially, using a configurable block size (1 MB is the default). Client requests are served as soon as the data becomes available. If a client requests data from parts of the file that have not been pre-fetched yet the proxy puts this request to the beginning of its download queue so as to serve the client with minimal latency. Vector reads are also fully supported. If a file is closed before pre-fetching is complete, further downloading is also stopped. When downloading of the file is complete it could in principle be moved to local storage. Currently, however, there are no provisions in the proxy itself to coordinate this procedure.

A state information file is maintained in parallel with each cached file to store the block size used for the file and a bit-field of blocks that have been committed to disk; this allows for complete cache recovery in case of a forced restart. Information about all file-accesses through the proxy (open & close time, number of bytes read and number of requests) is also put into the state file to provide cache reclamation algorithms with ample details about file usage.

The second mode supports on-demand downloading of individual blocks of a file (block-size can also be passes as cgi information in the file-open request). The distinguishing feature of the second implementation is that it only downloads the requested fixed-size blocks of a file.

While the main motivation is to provide pre-fetching of **HDFS** blocks (typical size 64 or 128 MB) when they become unavailable at the local site, either permanently or temporarily due to server overload or other transient failures. As **HDFS** block size is a per-file property, it has to be passed to the proxy on per-file basis as a **cgi** parameter on the file's URL Each block is stored as a separate file, post-fixed by block size and its offset in the full file; this facilitates potential reinjection back into **HDFS** to heal or increase replication of a file-block. Extensions to the **HDFS** have been developed to allow for an immediate fallback to **XRootD** access when local **HDFS** storage fails to provide the requested block.

That said, the mere fact that only referenced blocks are cached makes the caching proxy a powerful tool for any kind of remote file system, irrespective of block size. This is especially true in workloads that reference of a fraction of the file. Because only relevant blocks are cached, disk space usage is minimized and overall WAN performance is improved especially when running multiple jobs that largely reference the same blocks.

When additional file replicas exist in a data-federation, the remote data can be used to supplement local storage, to improve its robustness, and to provide a means for healing of local files. In particular, our intention is to avoid any local file replication of rarely-used, non-custodial data.

Unlike the full-file pre-fetching version, the partial-file proxy does not begin pre-fetching any data until a read request is actually received. At that point a check is made if the blocks required to fulfill the request exist on disk and, if they don't, they get queued for pre-fetching in whole. The client request is served as soon as the data becomes available.

3.1 Configuration

The disk caching proxy is implemented as a specialized caching plug-in that supplants the optional memory caching provided by the standard proxy server. Disk caching proxy specific directives and options are described below.

```
pss.cachelib path [libopts]          required directive
pss.ram bytes[m|g]                   required directive
pfc.blocksize bytes[k|m]
pfc.decisionlib path [libopts]
pfc.diskusage fracLow fracHigh
pfc.hdfsmode [hdfsbsize bytes[k|m]]
pfc.osslib path [libopts]
pfc.prefetch numPrefetchingBlocksPerFile
pfc.spaces data metadata
pfc.trace {none | error | warning | info | debug | dump}
pfc.user username
```

Required Directives

cachelib *path* [*libopts*]

specifies the path of he shared library than contains the disk caching proxy plug-in. It may be followed by options, if any. Currently, there are no options.

ram *bytes*[m|**g**]

maximum allowed RAM usage for caching proxy buffers that still need to be written to disk. Beyond that point the cache will serve further read requests by forwarding them to the remote server (note, it still needs to allocate buffers for this).

Optional Directives

blocksize *bytes*[**k**|**m**]

sets the block size used by proxy. Specify for *bytes* the size of a block. The quantity may be suffixed by **k** or **m** to indicate **k**ilo- or **m**egabytes, respectively. All read requests, including prefetching are rounded up into requests of this size and aligned to block boundaries. Read requests that cross one or more block boundaries are issued as separate read requests. This is also the size that gets written to disk in a single operation. The default is 1m (i.e. one megabyte).

decisionlib *path* [*libopts*]

specifies the location of the shared library that contains the cache decision plugin which determines whether or not to cache a file. All files are cached if a plugin is not specified. Specify for *path* the location of the decision plugin shared library and, optionally, plugin specific options for *libopts*.

diskusage *fracLow fracHigh*

specified the low and high watermark that starts and suspends purging of cached blocks to free up disk space. Specify for *fracLow* the space utilization percentage that triggers purging. Specify for *fracHigh* the percentage of space utilization that must be reached for purging to stop. The default is "0.9 0.95" (i.e. 90% 95%). Diskusage boundaries, can be specified also in **g** or **t** bytes units for **g**iga- and **t**era-bytes, respectively..

hdfsmode [***hdfsbsize*** *bytes* [**k**|**m**]]

enables storage of file fragments as separate files. This is specifically used for storage healing in HDFS. Specifying *bytes* sets the default fragment size. Specify for *bytes* the size of a fragment. The quantity may be suffixed by **k** or **m** to indicate **k**ilo- or **m**egabytes, respectively. The default is 128MB.

osslib *path* [*libopts*]

specifies the location of the shared library that contains the storage system plugin that should be used for all cache operations (i.e. read, write, etc). Specify for *path* the shared library that contains the plugin and, optionally, plugin specific parameters for *libopts*. The default is to use standard oss plug-in for all file system functions. Refer to the "Ofs/Oss Reference" on how to configure the standard oss plug-in.

prefetch *num*

maximum number of pre-fetching blocks per file. The default is 10. Set this value to 0 to disable pre-fetching.

spaces *data metadata*

spacifies the names of **oss** space name for data and the **oss** space name for metadata. The **oss.space** directive is used to create spaces. Default value of both is public.

trace {**none** | **error** | **warning** | **info** | **debug** | **dump**}

Set severity level of caching proxy log messages. The levels are listed in increasing verboseness. The default value is warning..

user *username*

specifies the user name to pass to the **osslib** plug-in when accessing the disk cache. Specify a valid Unix user name for *username*. Normally, this should be the user name of the proxy daemon.

Notes

1) All features of the standard Open Storage System are available to the disk caching proxy. Perhaps the most essential feature is logical volume management that allows the concatenation of disk partitions into a single logical volume. Thus, you can easily extend the disk cache. See the **oss.space** directive in the "Open Storage System Configuration Reference". There are many other **oss** options that may also be helpful in extending the features of the disk caching proxy.

Examples

a) Enable proxy file prefetching:

```
pps.cachelib   libXrdFileCache.so
pfc.ram        100g
oss.localroot  /data/xrd-file-cache
pfc.blocksize  512k
pfc.prefetch   8
```

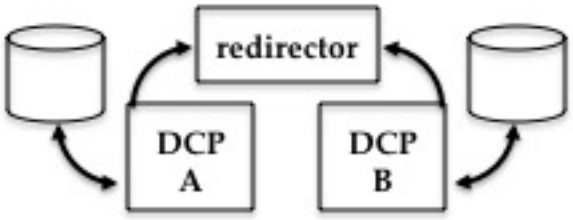
b) enable hdfs healing mode, with block size 64 MB:

```
pss.cachelib   libXrdFileCache.so
pfc.ram        100g
oss.localroot  /data/xrd-file-cache
pfc.hdfsmode   hdfsbsize 64m
```

3.2 Disk Caching Proxy Clusters

You may cluster disk-caching proxies (**DCP**'s) to provide for additional bandwidth and disk space using a redirector. However, clustering **DCP**'s is not the same as clustering standard proxy servers because **DCP**'s maintain a semi-permanent disk cache and cannot see the contents of each other's cache. Furthermore, clients opening a file must be vectored to the **DCP** that has already partially or fully cached the requested file and if the file is not present in any cache the client must be redirected to a **DCP** with sufficient disk space to cache the file.

In some sense a **DCP** is a combination of a proxy and a disk server. Hence, they need to be clustered as disk servers not as standard proxy servers. In the figure to the left, two disk caching proxy servers, **DCP-A** and **DCP-B** are clustered using a redirector. Each **DCP** has its own disk cache that may not be shared. In fact, any attempt to share the same space using, say, a distributed shared file system will result in data corruption or loss. The minimal configuration file for such a cluster is shown on the next page. The configuration file uses host names of **dcp-a**, **dcp-b**, and redirector for each server component in the cluster.



```
# Tell everyone who the manager is
#
all.manager redirector:1213

# The redirector and all cmsd's export /data red-only with the stage option. The stage
# option requests that if the file isn't found in the cluster the redirector should send
# the client to a PFC server with enough space to cache the file.
#
all.export /data stage r/o

# Configuration is different for the redirector, the server cmsd, and
# for the server xrootd. We break those out in the if-else-fi clauses.
#
if redirector

all.role manager

# Export with stage option - if the file isn't found in the cluster the
# redirector sends the client to a PFC server with enough free space.
#
all.export /data stage r/o

# Server's cmsd configuration – all PFC's are virtual data servers
#
else if exec cmsd

all.role server

# Export with stage option - this tells manager cmsd we can pull files from the origin
#
all.export /data stage r/o

# The cmsd uses the standard oss plug-in to locate files in the cache.
# oss.localroot directive should be the same as for the server.
#
oss.localroot /pfc-cache

# Server's xrootd configuration – all PFC's are virtual data servers
#
else

all.role server

# For xrootd, load the proxy plugin and the disk caching plugin.
#
ofs.osslib   libXrdPss.so
pss.cachelib libFileCache.so

# The server needs to write to disk, stage not relevant
#
all.export /data rw

# Tell the proxy where the data is coming from (arbitrary).
#
pss.origin someserver.domain.org:1094

# Tell the PFC's where the disk cache resides (arbitrary).
#
oss.localroot /pfc-cache

# Tell the PFC's available RAM
#
pfc.ram 100g

fi
```

4 Memory Caching Proxy

The memory caching proxy is almost identical to the standard proxy except that memory is used to hold recently read data. The memory cache is used to satisfy future reads if the requested data is present in the cache. This avoids re-fetching data from the originating server. A memory cache proxy may make LAN transfers more efficient, depending on the data access pattern.

You configure a memory caching proxy the same way you configure a standard proxy server with the addition of a **pss.cache** directive, described in the next section. Be aware that disk caching proxies and memory caching proxies are mutually exclusive (i.e. you cannot configure both in the same server).

4.1 Configuration

Proxy memory caching specific directives and options are described below.

```
pss.cache [cacheopts]
---------------------------------------------------------------------
cacheopts: [debug dbg] [logstats] [max2cache mc] [pagesize ps]
           [preread [prpgs [minrd]] [perf prf [rcalc]]] [r/w]
           [sfiles {off | on | .sfx}] [size sz] [cachepots]
```

Where:

cache

Configures memory caching of data. By default, memory caching is turned off. Specifying the **cache** directive with no *cachepots* is equivalent to specifying:

debug 0 pagesize 32k preread 0 sfiles off size 100m

If the **cachelib** directive is specified, the cache options have no effect. Otherwise, the following options may be specified:

debug sets the debugging level: 0 is off, 1 is low, 2 is medium, and 3 is high. The default is 0.

logstats prints statistics about cache usage for every file that is fully closed.

max2cache the largest read to cache in memory. The default is set to the **pagesize** value. The limit does not apply to pre-read operations. The *mc* value may be suffixed by **k**, **m**, or **g** to scale *mc* by 2^{10}, 2^{20}, or 2^{30}, respectively.

pagesize the size of each page in the cache. This also establishes the minimum read size from a data server. The *ps* value may be suffixed by **k**, **m**, or **g** to scale *ps* by 2^{10}, 2^{20}, or 2^{30}, respectively. The default is 32K.

preread enables pre-read operations. By default no data is pre-read. Specify the number of additional pages to be read past the last read page. You also follow *prpgs* by the read size that triggers a pre-read. Reads less than *minrd* cause a pre-read to occur. The *minrd* value may be suffixed by **k**, **m**, or **g** to scale *minrd* by 2^{10}, 2^{20}, or 2^{30}, respectively. The default is the **pagesize** value plus one. Specifying **preread** with no options is equivalent to specifying:

preread 1 perf 90

preread perf specifies the minimum required performance from automatic pre-reads and is part of the **preread** options. When this performance cannot be obtained for a file, pages are no longer automatically pre-read for the file. Performance is measured as the number pre-read pages used divided by the number actually pre-read times 100. Specify a value between 0 and 100 for *prf*. The default is 90 (i.e., 90% of all pre-reads must be useful). The *prf* may be followed by *rcalc*, the number of pre-read bytes that trigger a new performance calculation. The *rcalc* value may be suffixed by **k**, **m**, or **g** to scale *num* by 2^{10}, 2^{20}, or 2^{30}, respectively. The default is 52428800 (i.e. 50m).

r/w caches files opened in read/write mode. By default, only files opened read/only are cached.

sfiles enables or disables optimization for structured file (e.g., root files). Cache usage is optimized for the typical access patterns associated with structured files. Specify **off** to turn off this feature (the default), **on** to always use this feature, or a dot followed by a suffix (i.e. **.***sfx*). Only files whose names end with this suffix, including the dot, will have this optimization applied.

size specifies the size of the cache in bytes. The *sz* value may be suffixed by **k**, **m**, or **g** to scale *sz* by 2^{10}, 2^{20}, or 2^{30}, respectively. The default is 100m.

5 The netchk Utility

The **netchk** utility allows you to verify that an external client has a logical path to all of the required nodes in proxy configuration. The syntax is:

```
netchk { ssh | tcp } dest

dest:   [user@]host:port [dest]
```

Parameters

ssh only checks whether it is possible to ssh login through the *dest* host list.

tcp verifies that there is end-to-end message transitivity from the starting point through the *dest* host list.

user the username to use when doing an **ssh** login to each host in *dest*. The default is to use the current username.

host is a hostname or IP address that must be reachable. The first *host* in the list must be reachable from the starting node where the **netchk** command is issued. Each subsequent *host* in the *dest* list must be reachable from the previous *host* in the *dest* list.

port the port number where messages must be sent to. This is typically the port number for connects. This is only meaningful with **tcp** and the **ssh** parameter ignores the *port* specification.

Notes

1) The starting node must have **netchk** available. No other node need have **netchk** installed as the utility automatically but temporarily propagates itself across all of the hosts in *dest*.
2) All nodes, including the issuing node, must have **perl** installed along with the **IO::Socket** and **IPC::Open2** packages.
3) All hosts in *dest* must have **ssh** installed and the command issuer (or specified *user*) must have login access to each host in *dest*.
4) The **netchk** utility can be found in the **utils** directory of the **xrootd** distribution or in the add-ons section of the **xrootd.org** main page.

5) You should run **netchk** with the **ssh** parameter first to make sure there is a clear ssh login path to all hosts in *dest*.
6) Since the goal of **netchk** is to make sure there is message transitivity through all the listed hosts in *dest* using the ports that will be used by various daemons, the listed port must be available for use. This essentially means that the daemons that would normally use these ports must not be running on he hosts listed in *dest* when **netchk** is started.
7) A full transitivity test consists of trying paths between an external public node and all possible end-points.

Example

The following example tests whether there is appropriate transitivity between the **xrootd** proxy server to be run at z.domain.edu and the redirector to be run at x.domain.edu. Both servers will be using port 1024.

```
netchk tcp z.domain.edu:1094 x.domain.edu:1024
```

The test will make sure that the issuing node (which should be an external public node) can connect to z.domain.edu:1024 and that z.domain.edu:1024 can connect to x.domain.edu:1024. Two-way messages are sent across the connections to make sure they are not blocked.

6 Document Change History

16 Jun 2014

- Split description of proxy services from the ofs/oss manual. Hence, this is a new manual.
- Document the caching proxy service.

29 Jul 2014

- Document forwarding proxies.
- Document the **pss.origin** directive that configures a forwarding proxy.

5 Aug 2014

- Document the **pss.permit** directive that restricts outgoing connections of a forwarding proxy.

2 Dec 2014

- Document the disk caching proxy configuration directives (i.e. the ones that start with **pfc**).

14 Feb 2015

- Change the "**pfc.hdfs**" prefixed directives to "**pfc.file**" prefixed directives.
- Remove the **pfc.chachedir** directive.

11 Oct 2015

- Clarify the meaning of the high and low watermarks in the **pfc.diskusage** directive .

11 Jan 2017

- Document version 2 of the disk caching proxy which includes new and changed directives.