HDFS Proxy Guide

Table of contents

1 Introduction

HDFS Proxy is a proxy server through which a hadoop client (through HSFTP) or a standard HTTPS client (wget, curl, etc) can talk to a hadoop server and more importantly pull data from the sever. It put an access control layer in front of hadoop namenode server and extends its functionalities to allow hadoop cross-version data transfer.

2 Goals and Use Cases

2.1 Data Transfer from HDFS clusters

- User uses HSFTP protocol (hadoop distcp/fs, etc) to access HDFS proxy to copy out data stored on one or more HDFS clusters.
- User uses HTTPS protocol (curl, wget, etc) to access HDFS proxy to copy out data stored on one or more HDFS clusters

2.2 Cross-version Data Transfer

There are multiple HDFS clusters and possibly in different hadoop versions, each holding different data. A client need to access these data in a standard way without worrying about version compatibility issues.

2.3 User Access Control

- User Access Control through SSL certificates
- User Access Control through LDAP (Lightweight Directory Access Protocol) server

3 Comparison with NameNode's H(S)FTP Interface

NameNode has a http listener started at `dfs.namenode.http-address` with default port 50070 when NameNode is started and it provided a HFTP interface for the client. Also it could have a https listener started at `dfs.namenode.https-address` if `dfs.https.enable` is defined as true (by default, `dfs.https.enable` is not defined) to provide a HSFTP interface for client.

3.1 Advantages of Proxy Over NameNode HTTP(S) server

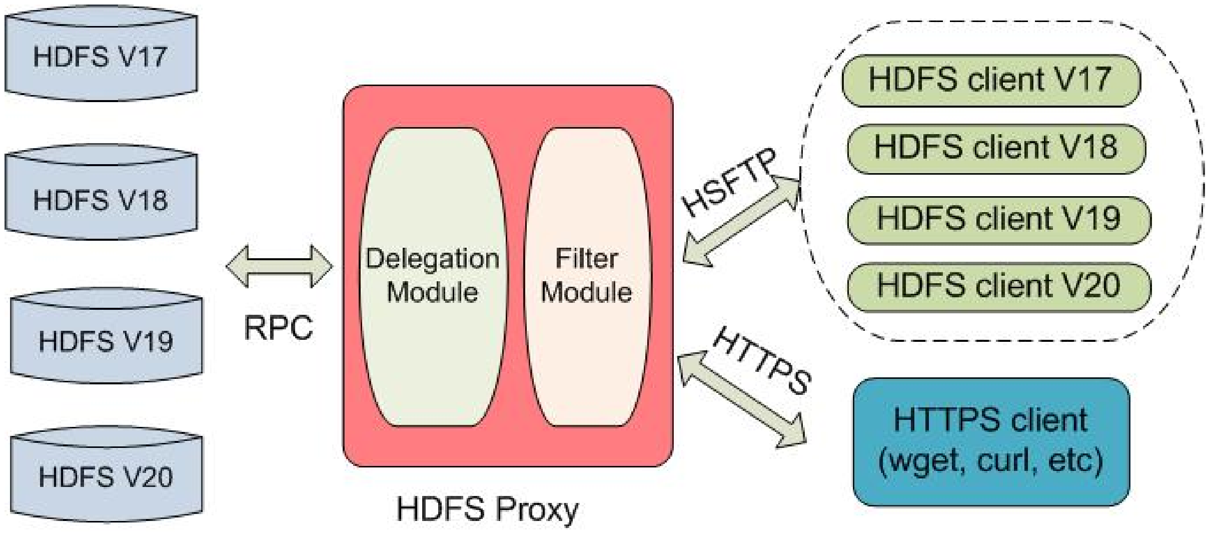
1. We can centralize access control layer to the proxy part so that NameNode server can lower its burden. In this sense, HDFS proxy plays a filtering role to control data access to NameNode and DataNodes. It is especially useful if the HDFS system has some sensitive data in it.
2. After modulizing HDFS proxy into a standalone package, we can decouple the complexity of HDFS system and expand the proxy functionalities without worring about affecting other HDFS system features.

3.2 Disadvantages of Using Proxy Instead of Getting Data Directly from H(S)FTP Interface: Slower in speed. This is due to

1. HDFS proxy need to first copy data from source cluster, then transfer the data out to the client.
2. Unlike H(S)FTP interface, where file status listing, etc., is through NameNode server, and real data transfer is redirected to the real DataNode server, all data transfer under HDFS proxy is through the proxy server.

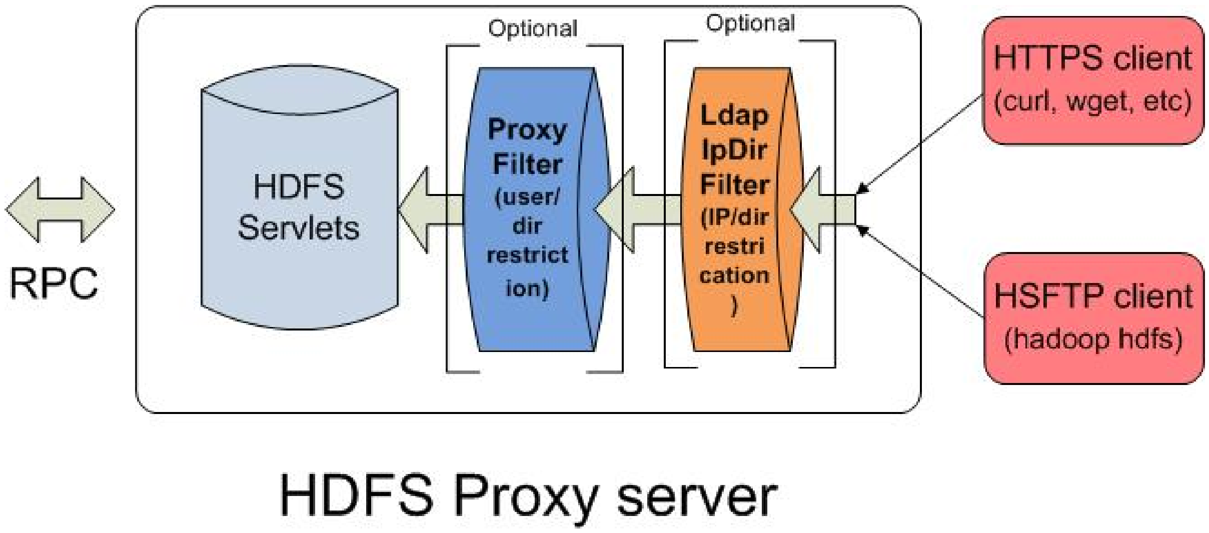
4 Design

4.1 Design Overview



As shown in the above figure, in the client-side, proxy server will accept requests from HSFTP client and HTTPS client. The requests will pass through a filter module (containing one or more filters) for access control checking. Then the requests will go through a delegation module, whose responsibility is to direct the requests to the right client version for accessing the source cluster. After that, the delegated client will talk to the source cluster server through RPC protocol using servlets.

4.2 Filter Module: Proxy Authentication and Access Control



To realize proxy authentication and access control, we used a servlet filter. The filter module is very flexible, it can be installed or disabled by simply changing the corresponding items in deployment descriptor file (web.xml). We implemented two filters in the proxy code: ProxyFilter and LdapIpDirFilter. The process of how each filter works is listed as below.

4.2.1 SSL certificate-based proxyFilter

1. A user will use a pre-issued SSL certificate to access the proxy.
2. The proxy server will authenticate the user certificate.
3. The user’s authenticated identity (extracted from the user’s SSL certificate) is used to check access to data on the proxy.
4. User access information is stored in two configuration files, user-certs.xml and user-permissions.xml.
5. The proxy will forward the user’s authenticated identity to HDFS clusters for HDFS file permission checking

4.2.2 LDAP-based LdapIpDirFilter

1. A standalone LDAP server need to be set-up to store user information as entries, and each entry contains userId, user group, IP address(es), allowable HDFS directories, etc.
2. An LDAP entry may contain multiple IP addresses with the same userId and group attribute to realize headless account.
3. Upon receiving a request, the proxy server will extract the user's Ip adress from the request header, query the LDAP server with the IP address to get the direcotry permission information, then compare that with the user request path to make a allow/deny decision.

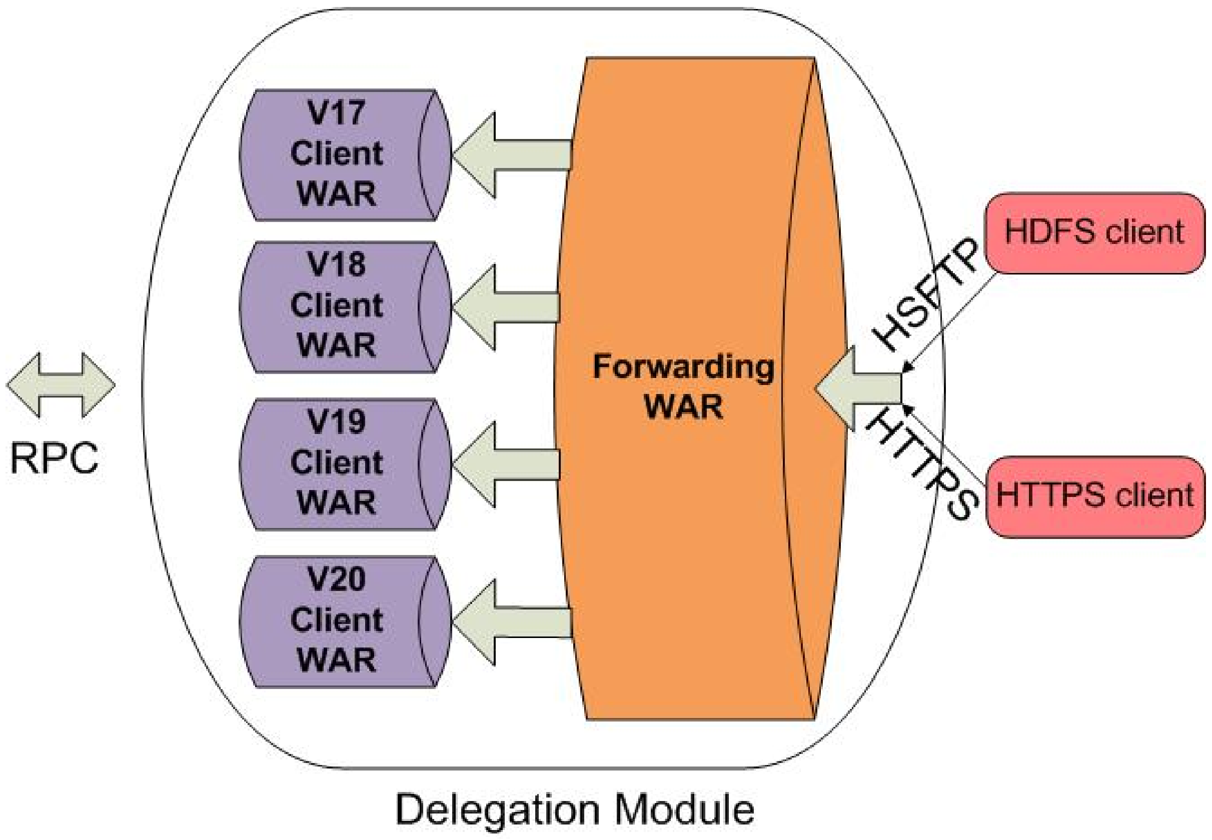
SSL-based proxyFilter provides strong PKI authentication and encryption, proxy server can create a self-signed CA using OpenSSL and use that CA to sign and issue certificates to clients.

Managing access information through configuration files is a convenient way to start and easy to set-up for a small user group. However, to scale to a large user group and to handle account management operations such as add, delete, and change access, a separate package or a different mechanism like LDAP server is needed.

The schema for the entry attributes in the LDAP server should match what is used in the proxy. The schema that is currently used in proxy is configurable through hdfsproxy-default.xml, but the attributes should always contain IP address (default as uniqueMember), userId (default as uid), user group (default as userClass), and alloable HDFS directories (default as documentLocation).

Users can also write their own filters to plug in the filter chain to realize extended functionalities.

4.3 Delegation Module: HDFS Cross-version Data Transfer



As shown in the Figure, the delegation module contains two parts:

1. A Forwarding war, which plays the role of identifying the requests and directing the requests to the right HDFS client RPC version.
2. Several RPC client versions necessary to talk to all the HDFS source cluster servers.

All servlets are packaged in the WAR files.

Strictly speaking, HDFS proxy does not by itself solve HDFS cross-version communication problem. However, through wrapping all the RPC client versions and delegating the client requests to the right version of RPC clients, HDFS proxy functions as if it can talk to multiple source clusters in different hadoop versions.

Packaging the servlets in the WAR files has several advantages:

1. It reduces the complexity of writing our own ClassLoaders for different RPC clients. Servlet container (Tomcat) already uses separate ClassLoaders for different WAR files.
2. Packaging is done by the Servlet container (Tomcat). For each client WAR file, its Servlets only need to worry about its own version of source HDFS clusters.

Note that the inter-communication between servlets in the forwarding war and that in the specific client version war can only be through built-in data types such as int, String, etc, as such data types are loaded first through common classloader.

4.4 Servlets: Where Data transfer Occurs

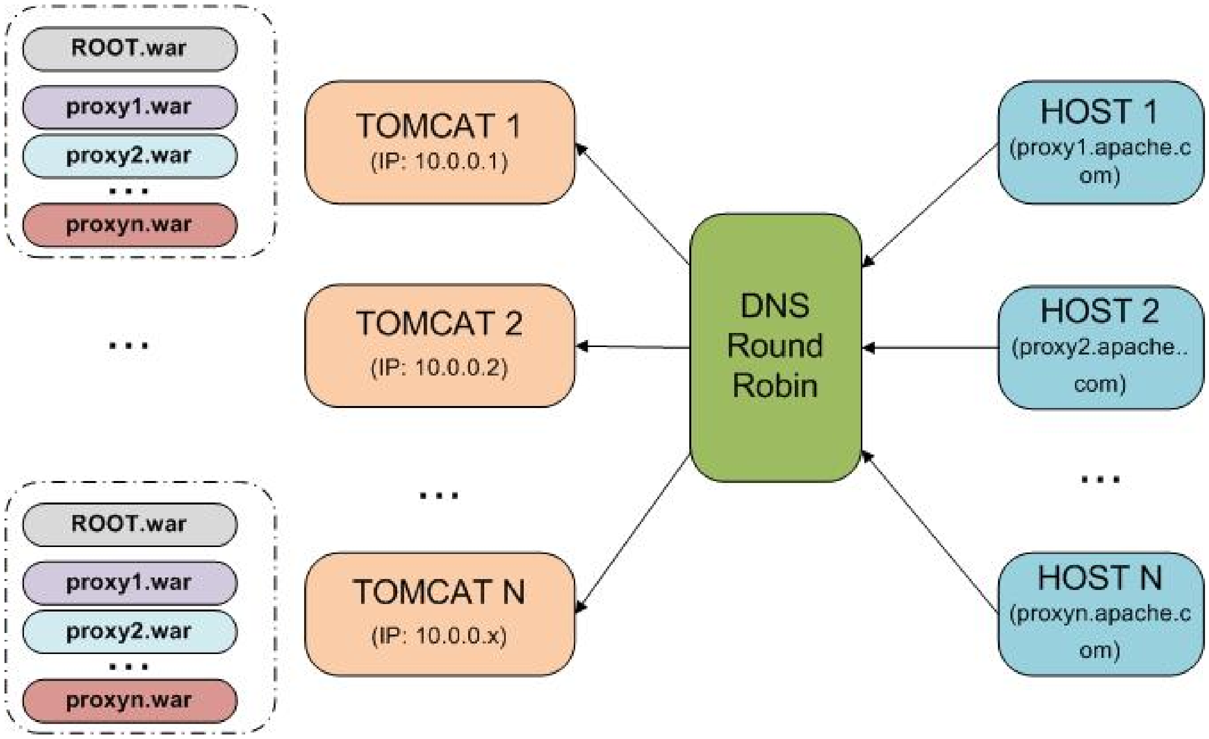
Proxy server functionality is implemented using servlets deployed under servlet container. Specifically, there are 3 proxy servlets `ProxyListPathsServlet`, `ProxyFileDataServlet`, and `ProxyStreamFile`. Together, they implement the same H(S)FTP interface as the original `ListPathsServlet`, `FileDataServlet`, and `StreamFile` servlets do on an HDFS cluster. In fact, the proxy servlets are subclasses of the original servlets with minor changes like retrieving client UGI from the proxy server, etc. All these three servlets are put into the client war files.

The forwarding proxy, which was implemented through `ProxyForwardServlet`, is put in a separate web application (ROOT.war). All client requests should be sent to the forwarding proxy. The forwarding proxy does not implement any functionality by itself. Instead, it simply forwards client requests to the right web applications with the right servlet paths.

Forwarding servlets forward requests to servlets in the right web applications through servlet cross-context communication by setting `crossContext="true"` in servlet container's configuration file

Proxy server will install a servlet, `ProxyFileForward`, which is a subclass of `ProxyForwardServlet`, on path /file, which exposes a simple HTTPS GET interface (internally delegates the work to `ProxyStreamFile` servlet via forwarding mechanism discussed above). This interface supports standard HTTP clients like curl, wget, etc. HTTPS client requests on the wire should look like `https://proxy_address/file/file_path`

4.5 Load Balancing and Identifying Requests through Domain Names



The delegation module relies on the forwarding WAR to be able to identify the requests so that it can direct the requests to the right HDFS client RPC versions. Identifying the requests through Domain Name, which can be extracted from the request header, is a straightforward way. Note that Domain Name can have many alias through CNAME. By exploiting such a feature, we can create a Domain Name, then create many alias of this domain name, and finally make these alias correspond to different client RPC request versions. As the same time, we may need many servers to do load balancing. We can make all these servers (with different IP addresses) point to the same Domain Name in a Round-robin fashion. By doing this, we can realize default load-balancing if we have multiple through proxy servers running in the back-end.

5 Jetty-based Installation and Configuration

With Jetty-based installation, only part of proxy features are supported.

5.1 Supporting Features

- Single Hadoop source cluster data transfer
- Single Hadoop version data transfer
- Authenticate users via user SSL certificates with `ProxyFilter` installed
- Enforce access control based on configuration files.

5.2 Configuration Files

1. **hdfsproxy-default.xml**

Name	Description
hdfsproxy.https.address	the SSL port that hdfsproxy listens on.
hdfsproxy.hosts	location of hdfsproxy-hosts file.
hdfsproxy.dfs.namenode.address	namenode address of the HDFS cluster being proxied.
hdfsproxy.https.server.keystore.resource	location of the resource from which ssl server keystore information will be extracted.
hdfsproxy.user.permissions.file.location	location of the user permissions file.
hdfsproxy.user.certs.file.location	location of the user certs file.
hdfsproxy.ugi.cache.ugi.lifetime	The lifetime (in minutes) of a cached ugi.

2. **ssl-server.xml**

Name	Description
ssl.server.truststore.location	location of the truststore.
ssl.server.truststore.password	truststore password.
ssl.server.keystore.location	location of the keystore.
ssl.server.keystore.password	keystore password.
ssl.server.keystore.keypassword	key password.

3. **user-certs.xml**

Name	Description
This file defines the mappings from username to comma seperated list of certificate serial numbers that the user is allowed to use. One mapping per user. Wildcard characters, such as "*" and "?", are not recognized. Any leading or trailing whitespaces are stripped/ignored. In order for a user to be able to do "clearUgiCache" and "reloadPermFiles" command, the certification serial number he use must also belong to the user "Admin".	

4. **user-permissions.xml**

Name	Description
This file defines the mappings from user name to comma seperated list of directories/files that the user is allowed to access. One mapping per user. Wildcard characters, such as "*" and "?", are not recognized. For example, to match "/output" directory, one can use "/output" or "/output/", but not "/output/*". Note that any leading or trailing whitespaces are stripped/ignored for the name field.	

5.3 Build Process

Under `$HADOOP_HDFS_HOME` do the following

```
$ ant clean tar
$ cd src/contrib/hdfsproxy/
$ ant clean tar
```

The `hdfsproxy-*.tar.gz` file will be generated under `$HADOOP_HDFS_HOME/build/contrib/hdfsproxy/`. Use this tar ball to proceed for the server start-up/shutdown process after necessary configuration.

5.4 Server Start up and Shutdown

Starting up a Jetty-based HDFS Proxy server is similar to starting up an HDFS cluster. Simply run `hdfsproxy` shell command. The main configuration file is `hdfsproxy-default.xml`, which should be on the classpath. `hdfsproxy-env.sh` can be used to set up environmental variables. In particular, `JAVA_HOME` should be set. As listed above, additional configuration files include `user-certs.xml`, `user-permissions.xml` and `ssl-server.xml`, which are used to specify allowed user certs, allowed directories/files, and ssl keystore information for the proxy, respectively. The location of these files can be specified in `hdfsproxy-default.xml`. Environmental variable `HDFSPROXY_CONF_DIR` can be used to point to the directory where these configuration files are located. The configuration files (`hadoop-site.xml`, or `core-site.xml` and `hdfs-site.xml`) of the proxied HDFS cluster should also be available on the classpath .

Mirroring those used in HDFS, a few shell scripts are provided to start and stop a group of proxy servers. The hosts to run hdfsproxy on are specified in `hdfsproxy-hosts` file, one host per line. All hdfsproxy servers are stateless and run independently from each other.

To start a group of proxy servers, do

```
$ start-hdfsproxy.sh
```

To stop a group of proxy servers, do

```
$ stop-hdfsproxy.sh
```

To trigger reloading of `user-certs.xml` and `user-permissions.xml` files on all proxy servers listed in the `hdfsproxy-hosts` file, do

```
$ hdfsproxy -reloadPermFiles
```

To clear the UGI caches on all proxy servers, do

```
$ hdfsproxy -clearUgiCache
```

5.5 Verification

Use HSFTP client

```
bin/hadoop fs -ls "hsftp://proxy.address:port/"
```

6 Tomcat-based Installation and Configuration

With tomcat-based installation, all HDFS Proxy features are supported

6.1 Supporting Features

- Multiple Hadoop source cluster data transfer
- Multiple Hadoop version data transfer
- Authenticate users via user SSL certificates with `ProxyFilter` installed
- Authentication and authorization via LDAP with `LdapIpDirFilter` installed
- Access control based on configuration files if `ProxyFilter` is installed.
- Access control based on LDAP entries if `LdapIpDirFilter` is installed.
- Standard HTTPS Get Support for file transfer

6.2 Source Cluster Related Configuration

1. **hdfsproxy-default.xml**

Name	Description
fs.defaultFS	Source Cluster NameNode address
dfs.blocksize	The block size for file tranfers
io.file.buffer.size	The size of buffer for use in sequence files. The size of this buffer should probably be a multiple of hardware page size (4096 on Intel x86), and it determines how much data is buffered during read and write operations

6.3 SSL Related Configuration

1. **hdfsproxy-default.xml**

Name	Description
hdfsproxy.user.permissions.file.location	location of the user permissions file.

Name	Description
hdfsproxy.user.certs.file.location	location of the user certs file.
hdfsproxy.ugi.cache.ugi.lifetime	The lifetime (in minutes) of a cached ugi.

2. **user-certs.xml**

Name	Description
This file defines the mappings from username to comma seperated list of certificate serial numbers that the user is allowed to use. One mapping per user. Wildcard characters, such as "*" and "?", are not recognized. Any leading or trailing whitespaces are stripped/ignored. In order for a user to be able to do "clearUgiCache" and "reloadPermFiles" command, the certification serial number he use must also belong to the user "Admin".	

3. **user-permissions.xml**

Name	Description
This file defines the mappings from user name to comma seperated list of directories/files that the user is allowed to access. One mapping per user. Wildcard characters, such as "*" and "?", are not recognized. For example, to match "/output" directory, one can use "/output" or "/output/", but not "/output/*". Note that any leading or trailing whitespaces are stripped/ignored for the name field.	

6.4 LDAP Related Configuration

1. **hdfsproxy-default.xml**

Name	Description
hdfsproxy.ldap.initial.context.factory	LDAP context factory.
hdfsproxy.ldap.provider.url	LDAP server address.
hdfsproxy.ldap.role.base	LDAP role base.

6.5 Tomcat Server Related Configuration

1. **tomcat-forward-web.xml**

Name	Description
This deployment descritor file defines how servlets and filters are installed in the forwarding war (ROOT.war). The default filter installed is `LdapIpDirFilter`, you can change to `ProxyFilter` with `org.apache.hadoop.hdfsproxy.ProxyFilter` as you `filter-class`.	

2. **tomcat-web.xml**

Name	Description
	This deployment descritor file defines how servlets and filters are installed in the client war. The default filter installed is `LdapIpDirFilter`, you can change to `ProxyFilter` with `org.apache.hadoop.hdfsproxy.ProxyFilter` as you `filter-class`.

3. **$TOMCAT_HOME/conf/server.xml**

Name	Description
	You need to change Tomcat's server.xml file under $TOMCAT_HOME/conf as detailed in tomcat 6 ssl-howto. Set `clientAuth="true"` if you need to authenticate client.

4. **$TOMCAT_HOME/conf/context.xml**

Name	Description
	You need to change Tomcat's context.xml file under $TOMCAT_HOME/conf by adding `crossContext="true"` after `Context`.

6.6 Build and Deployment Process

6.6.1 Build forwarding war (ROOT.war)

Suppose hdfsproxy-default.xml has been properly configured and it is under ${user.home}/proxy-root-conf dir. Under `$HADOOP_HDFS_HOME` do the following

```
$ export HDFSPROXY_CONF_DIR=${user.home}/proxy-root-conf
$ ant clean tar
$ cd src/contrib/hdfsproxy/
$ ant clean forward
```

The `hdfsproxy-forward-*.war` file will be generated under `$HADOOP_HDFS_HOME/build/contrib/hdfsproxy/`. Copy this war file to tomcat's webapps directory and rename it at ROOT.war (if ROOT dir already exists, remove it first) for deployment.

6.6.2 Build cluster client war (client.war)

Suppose hdfsproxy-default.xml has been properly configured and it is under ${user.home}/proxy-client-conf dir. Under `$HADOOP_HDFS_HOME` do the following

```
$ export HDFSPROXY_CONF_DIR=${user.home}/proxy-client-conf
$ ant clean tar
$ cd src/contrib/hdfsproxy/
$ ant clean war
```

The `hdfsproxy-*.war` file will be generated under `$HADOOP_HDFS_HOME/build/contrib/hdfsproxy/`. Copy this war file to tomcat's webapps directory and rename it properly for deployment.

6.6.3 Handle Multiple Source Clusters

To proxy for multiple source clusters, you need to do the following:

1. Build multiple client war with different names and different hdfsproxy-default.xml configurations
2. Make multiple alias using CNAME of the same Domain Name
3. Make sure the first part of the alias match the corresponding client war file name. For example, you have two source clusters, sc1 and sc2, and you made two alias of the same domain name, proxy1.apache.org and proxy2.apache.org, then you need to name the client war file as proxy1.war and proxy2.war respectively for your deployment.

6.7 Server Start up and Shutdown

Starting up and shutting down Tomcat-based HDFS Proxy server is no more than starting up and shutting down tomcat server with tomcat's bin/startup.sh and bin/shutdown.sh script.

If you need to authenticate client certs, you need either set `truststoreFile` and `truststorePass` following tomcat 6 ssl-howto in the configuration stage or give the truststore location by doing the following

```
export JAVA_OPTS="-Djavax.net.ssl.trustStore=${user.home}/
truststore-location -
Djavax.net.ssl.trustStorePassword=trustpass"
```

before you start-up tomcat.

6.8 Verification

HTTPS client

```
curl -k "https://proxy.address:port/file/file-path"
wget --no-check-certificate "https://proxy.address:port/file/
file-path"
```

HADOOP client

```
bin/hadoop fs -ls "hsftp://proxy.address:port/"
```

7 Hadoop Client Configuration

- **ssl-client.xml**

Name	Description
ssl.client.do.not.authenticate.server	if true, trust all server certificates, like curl's -k option
ssl.client.truststore.location	Location of truststore
ssl.client.truststore.password	truststore password

Name	Description
ssl.client.truststore.type	truststore type
ssl.client.keystore.location	Location of keystore
ssl.client.keystore.password	keystore password
ssl.client.keystore.type	keystore type
ssl.client.keystore.keypassword	keystore key password
ssl.expiration.warn.days	server certificate expiration war days threshold, 0 means no warning should be issued