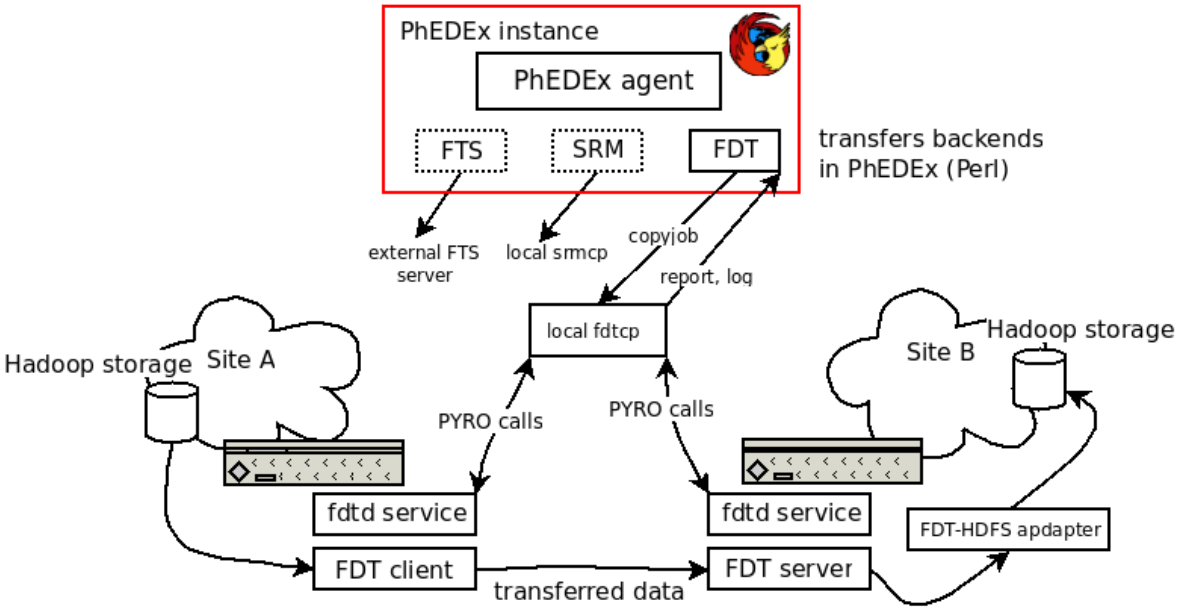
Table of Contents

Design of integration of FDT tool with PhEDEx.

The diagram of the intended integration outlines development of three new components to interface high-performance transfer tool FDT (Fast Data Transfer) with PhEDEx (CMS Data Transfers):

- **FDT** (FDT.pm) - Perl module, transfer backend in PhEDEx - source code located within PhEDEx installation
- **fdtcp** - Python wrapper, interface between PhEDEx and FDT data transfers, features:
  - ♦ prepares **copyjob/fileList** transfer file as required by FDT
  - ♦ does necessary translation of source, destination file names
  - ♦ harvests **report** and **log** files to propagate back to PhEDEx
  - ♦ invokes remote **fdtd** service (forwards certificate proxy for authentication)
- **fdtd** - FDT daemon, FDT service wrapper, features:
  - ♦ permanently running daemon on FDT-enabled sites
  - ♦ receives request (PYRO (Python Remote Objects) calls) to launch either FDT client party on the source sites or FDT server party on the destination sites. FDT writing server must use Hadoop HDSF-FDT adapter if destination storage backend is Hadoop (by FDT client, resp. server is meant FDT Java application (`fdt.jar`))
  - ♦ responsible for authentication.



The scenario assumes transfer of file(s) from site A to site B. Source site (A) **fdtd** invokes client (reading, source) party of the FDT application and destination site (B) **fdtd** invokes server (writing, destination) party of the FDT application. FDT application (Java) is invoked only on demand and the process is closed once the transfer is over.

Both `fdtcp` and `fdtd` will be written in Python. Communication between `fdtcp` and `fdtd` will utilise PYRO (Python Remote Objects).

The envisaged feature is on-the-fly transfer monitoring. For this may be worth considering RESTful webservice API by means of CherryPy web framework so that the monitoring could be queried from web browser as well.

Active monitoring utilising MonALISA `ApMon` will be used throughout the transfer stream from PhEDEx transfer backend `FDT.pm` to FDT Java application.

Status 2010-03-04 - fdtcp/fdtd functional (yet with no Grid authentication)

Source codes available in this repository phedex-fdt

Using latest FDT 0.9.10

Third-party transfers implemented via `fdtcp.py` and `fdtd.py`, tested launching `fdtcp.py` from `t2-headnode.ultralight.org`:

```
fdtcp.py t3-susy.ultralight.org:/mnt/hadoop/user/maxa/1GB.test gridftp01.ultralight.org:/tmp/1GB.
```

Group transfers utilizing batch transfer file, resp. `copyjobfile` as known from `srmcp`, implemented. Various source host, destination host pairs are broken down to corresponding third party transfers, the same source, destination pairs result into `group transfer` utilizing a single FDT Java client/server pair.

```
fdtcp.py --copyjobfile=./copyjobfile
```

example of `copyjobfile`:

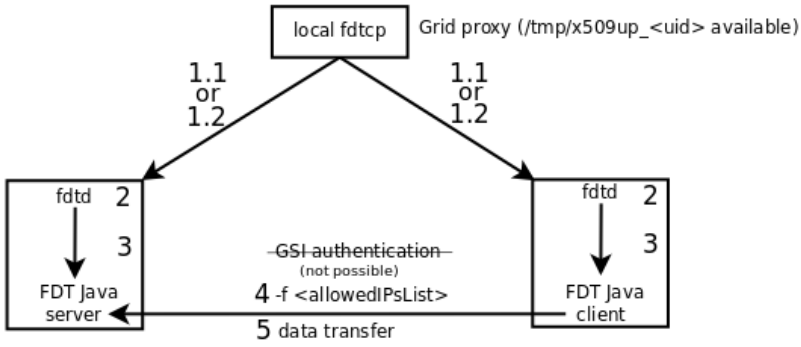
```
t3-susy.ultralight.org:/mnt/hadoop/user/maxa/1GB-01.test gridftp01.ultralight.org:/tmp/1GB-01.tes
t3-susy.ultralight.org:/mnt/hadoop/user/maxa/1GB-02.test gridftp02.ultralight.org:/tmp/1GB-02.tes
t3-susy.ultralight.org:/mnt/hadoop/user/maxa/1GB-03.test gridftp03.ultralight.org:/tmp/1GB-03.tes
t3-susy.ultralight.org:/mnt/hadoop/user/maxa/1GB-04.test gridftp04.ultralight.org:/tmp/1GB-04.tes
```

The main issue is to implement **Grid authentication** and eventually adding MonALISA `ApMon` monitoring.

Producing `report` file with transfers results is not yet implemented. It is not functionally required by PhEDEx which carries out it own checks and `report` file content is only published on the web interface for debugging purposes.

The plan for **Grid authentication** is that `fdtd` instance will have some `host certificate`, `host key` available for invocation of FDT server instance using GSI authentication as described here. FDT client instance will be launched with `user key`, `user certificate` as forwarded by the user launching `fdtcp.py` job. User mapping at the destination site will be carried out using GUMS - Grid User Management System service. Final file ownership change will be done via a simple sudo-run external script so that neither `fdtd.py` nor `FDT Java` don't have to run with root privileges.

Status 2010-03-15 - Grid authentication outline



The user of `fdtcp.py` (or PhEDEx agent) has its Grid proxy certificate ready in the `/tmp/x509up_uUID` file

1. Authentication (and encryption of the initial handshake) between `fdtcp.py` and remote `fdtd.py` by user's proxy: two options considered so far (only one of them will be used):
    1. Utilize GSI Python infrastructure pyGlobus, PyGridWare and pyGsi
        - ◊ download links non functional, e.g. Not Found, Not Found, Not Found, Not Found ; correction: pyGlobus is included in the GT releases, the latest GT release containing pyGlobus is 4.0.7 (March 2008), there has been no update on pyGlobus since, the latest GT release is 5.0.0 (January 2010)
        - ◊ dependency on pieces of Twisted and Zope
        - ◊ PyGridWare authenticates proxy against grid map file, no support for GUMS
    2. Use PYRO/SSL support for secure transport of user's proxy to remote `fdtd.py`
        - ◊ both `fdtcp.py` and `fdtd.py` require a key and certificate issued by a trusted CA
        - ◊ no further dependency packages required apart from SSL - M2Crypto
        - ◊ PYRO is already used for `fdtcp.py`, `fdtd.py` communication
2. At this point the user's proxy is authenticated against grid map file (the same in both sub-scenarios at the previous point) and `fdtd.py` finds the local Grid user matching remote user's proxy certificate (mapping)
3. `fdtd.py` issues either client party or server party of FDT under local Grid user via `sudo`
4. Both FDT client, server perform Grid authentication (GSI) - server is started with its pair of service key, certificate and the client is supplied user's proxy certificate
5. The authorization is secured by the local Grid user rights, i.e. reading from the specified location at the FDT client side, resp. writing at the FDT server side. Finally - the actual data transfer begins, files arrive under correct ownership

Grid authentication solution

- Using SSL directly in PYRO is possibly the easiest and transparent solution, but requires creating additional SSL key, certificate pair for `fdtcp.py` client - this may not be acceptable in the CMS/Grid environment
- Using Grid proxy certificate with PYRO/SSL doesn't work, certificate doesn't verify
- Laborious experiments with `pyGlobus`, `pyGridWare` showed the libraries are rather obsolete, and experiments to set up GSI authentication and secure encrypted communication weren't successful
- Final solution to Grid authentication is to develop a pair `AuthClient`, `AuthService` - simple GSI authentication applications in Java, taking GSI authentication solution completely from FDT and using this pair to safe transfer of user's proxy to the `fdtd.py` remote host
- It turned out that `point 4` can't be accomplished since at that stage FDT Java server runs under local grid user account (unprivileged) and as such can't have access to the grid credentials (service key and certificate). So, instead of this (secondary) GSI authentication at this stage it is used `-f <allowedIPsList>` mechanism of the FDT Java server allowing only certain client IP to connect to itself. This is considered strong enough and in fact similar to how `ssh` mode in FDT works anyway.

-- ZdenekMaxa - 01-Feb-2010

This topic: Main > PhEDExFDTIntegration
Topic revision: r9 - 14-Jul-2010 - 11:56:11 - ZdenekMaxa