"PROOF on Demand" (PoD) - v3.16

Anar Manafov, GSI, Scientific Computing division <A.Manafov@gsi.de>

"PROOF on Demand" (PoD) - v3.16

by Anar Manafov

Copyright © 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014 GSI Helmholtzzentrum für Schwerionenforschung GmbH, Scientific Computing division

Table of Contents

List of Figures

List of Tables

List of Examples

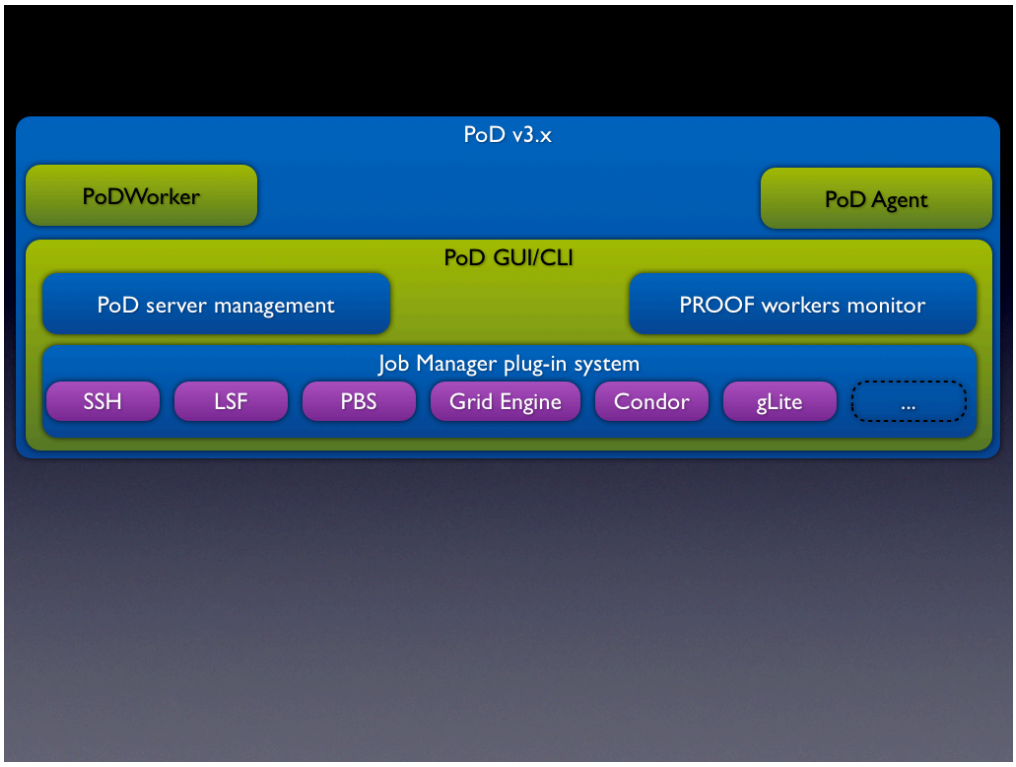
1. Introduction

1.1. PROOF on Demand

PROOF on Demand (PoD) is a tool-set (see Figure 1.1, “A generic schema of PoD”) developed at GSI, which sets up a PROOF cluster on any resource management system. PoD is a user oriented product with an easy to use GUI and a command-line interface. It is fully automated. No administrative privileges or special knowledge is required to use it. PoD gives users, who don't have a centrally-administrated static PROOF cluster at their institutions, the possibility to enjoy the full power of interactive analysis with PROOF.

PoD is a specially designed solution to provide a PROOF cluster on the fly.

Figure 1.1. A generic schema of PoD



1.2. Features

- **Easy to use**

  The process of installation is very simple and fully automated. PoD works out of the box. Its distribution contains preconfigured modules and everything users need to just immediately start to work with it right after the installation.

- **GUI & Command-line**

  PoD provides a simple and intuitive graphics user interface in order to simplify access to its functionality. For user's convenience there is also a command line interface, it helps to manage a PoD cluster remotely or use it in a batch mode.

- **Native PROOF connections**

Whenever possible, PoD setups direct PROOF connections between nodes. It results in a full functional PROOF cluster. Users get native speed and the whole range of PROOF features. To use native connections an incoming traffic must be allowed on PoD workers for a defined port. Otherwise PoD uses packet-forwarding algorithms.

- **Packet-forwarding**

  When worker nodes are behind a firewall then PoD uses its packet-forwarding algorithms to maintain the PROOF traffic. The algorithms are very efficient, there will be no speed penalty, but some PROOF functions are limited.

- **Multiuser/-core environment**

  PoD implements automatic port mapping algorithms to properly handle cases when several users start PoD instances (servers/ workers) on the same machine. PoD also automatically manages situations when multiple PoD workers are started on the same node. Private PoD instances can't disturb each other.

- **Different job managers**

  PoD supports different job managers via a plug-in system. It is a very easy to extend system. PoD is currently shipped with the following plug-ins:

  - SSH,
  - LSF (Load Sharing Facility),
  - PBS Pro/OpenPBS/Torque (Portable Batch System),
  - Grid Engine (Oracle/Sun Grid Engine),
  - Condor,
  - LoadLeveler (IBM Tivoli Workload Scheduler LoadLeveler),
  - gLite.

2. Requirements

2.1. User interface

PoD UI/Server/WN run on Linux and Mac OS X.

General requirements:

- Incoming connection on pod-agent's port (configurable)
- ROOT 5.25.04 or higher (xrootd enabled)
- BOOST 1.33.1 or higher (to use pod-remote(1), BOOST 1.41.0 or higher is required)
- cmake 2.6.2 or higher
- shell: BASH

Additional requirements for gLite plug-in:

- gLite UI 3.2
- gLite WMS (WMProxy endpoint)

Additional requirements for SSH plug-in:

- A public key access (or password less, via ssh-agend, for example) to destination worker nodes.

2.2. Workers

General requirements:

- Outgoing connection for a port range (configurable). This is needed by pod-agent worker to be able to connect to PoD server
- ROOT 5.25.04 or higher (xrootd enabled). A ROOT installation on worker nodes is not strictly required, but is recommended. If available, it will significantly speed up the start up time of the PoD workers. If there is no ROOT on WNs, PoD will download and use its default ROOT version for every worker node, which can increase the start up time of the workers and the network traffic.
- shell: BASH

Additional requirements for gLite plug-in:

- gLite WNs (at least v3.0)

3. Download

3.1. Download location

Please, use PoD's Download page to get the latest version.

3.2. PoD Version Number Scheme

PoD version has a form of MAJOR.MINOR(.PATCH), where:

- MAJOR - the major number is increased when there are significant jumps in functionality.
- MINOR - the minor number is incremented when only minor features or significant fixes have been added.
- PATCH - represents a number of commits (patches) to a current major.minor pair.

Note

Starting with version 3.4, PoD changes its version numbering. It reflects the fact that PoD is both a production system and a research project. PoD now uses odd minor version numbers to denote development releases and even minor version numbers to denote stable releases.

4. Installation

PoD supports Private and Shared installations.

A Private Installation - it is when a user installs PoD for individual use to his/her local folder. Any Private Installation can be used by other users as well. It's just a matter of file privileges.

A Shared Installation - it is when a site administrator installs PoD in some central location, so it can be shared by many users. This type of installation may be convenient for some users, since they don't need to install PoD by their own. In case of a shared Installation you need to execute one additional step, see Section 4.5, “Step #5: PoD shared Installation”. All the rest is the same as with Private Installations.

Be advised, that in both cases PoD acts identically and always provides private clusters, one for each user. In case of a shared installation, users share only binaries and configurations, but each user get's its own PoD instance and can't disturb other users. Each user can tune PoD by changing the PoD user defaults configuration in `$HOME/.PoD/PoD.cfg`.

4.1. Step #1: Unpack the source

Unpack PoD tarball:

```
tar -xzvf PoD-X.Y.Z-Source.tar.gz
```

Tar will created a new directory `./PoD-X.Y.Z-Source`, where *X.Y.Z* represents a version of PoD.

4.2. Step #2: Configure the source

Change to the PoD source directory:

```
cd ./PoD-X.Y.Z-Source
```

You can adjust some configuration settings in the `BuildSetup.cmake` bootstrap file. The following is a list of variables:

Table 4.1. PoD configuration variables

Variable	Description
CMAKE_INSTALL_PREFIX	Install path prefix, prepended onto install directories. (default $HOME/PoD/[PoD_Version])
CMAKE_BUILD_TYPE	Set cmake build type. Possible options are: None, Debug, Release, RelWithDebInfo, MinSizeRel (default Release)
BUILD_DOCUMENTATION	Build source code documentation. Possible options are: ON/OFF (default OFF)
BUILD_TESTS	Build PoD tests. Possible options are: ON/OFF (default OFF)

Now, prepare a build directory for an out-of-source build and configure the source:

```
```

```
mkdir build
cd build
cmake -C ../BuildSetup.cmake ..
```

Tip

If for some reason, for example a missing dependency, configuration failed. After you get the issue fixed, right before starting the **cmake** command it is recommended to delete everything in the build directory recursively. This will guaranty a clean build every time the source configuration is restarted.

4.3. Step #3: Build and install

Issue the following commands to build and install PoD:

```
make
make install
```

Installation Prefix

Please note, that by default PoD will be installed in `$HOME`/PoD/X.Y.Z, where X.Y.Z is a version of PoD. However users can change this behavior by setting the install prefix path in the bootstrap script `BuildSetup.cmake`. Just uncomment the setting of CMAKE_INSTALL_PREFIX variable and change dummy MY_PATH_HERE to a desired path.

4.4. Step #4: PoD Environment

In order to enable PoD's environment you need to source the `PoD_env.sh` script. Change to your newly installed PoD directory and issue:

```
cd [PoD INSTALL DIRECTORY]
source PoD_env.sh
```

You need to source this script every time before using PoD in a new system shell. Simplify it by sourcing the script in your bash profile.

Now the installation is done. But if you were preparing a shared installation, then please see the Section 4.5, “Step #5: PoD shared Installation” as well.

Important

If there were problems during the installation, please see Chapter 13, *Support* or Chapter 12, *Known Issues*.

4.5. Step #5: PoD shared Installation

If you installed PoD in some central location (by changing the default CMAKE_INSTALL_PREFIX, see Section 4.2, “Step #2: Configure the source”), than you need to make one simple additional step.

Normally central installations or shared installations are restricted for read-only for users. You therefore need to provide pre-compiled binaries for worker nodes, which are kept in `$POD_LOCATION`, so that other users could simple re-use them. To do that just issue the following command:

```
pod-server getbins
```

Next time (TO-DO) we will provide a documentation on how to prepare your own binaries for worker nodes.

5. Configuration

As it was mentioned above PoD consists of several modules, each module respects PoD user defaults settings. PoD is shipped with predefined configuration values, which should work in most use cases. However by changing PoD user defaults values, you can fine-tune PoD for a specific environment and needs. Recommended only for advanced users.

5.1. PoD user defaults configuration

Since PoD v2.1.1 a user defaults configuration file is supported. This is the configuration entry point of PoD. All modules configure themselves according to that file. It must be located either in the `$HOME/.PoD/` directory on in the `$POD_LOCATION/etc/` directory and be called `PoD.cfg`.

Tip

Every time users sources PoD's environment script, the script checks whether the configuration file is exists and creates it if it's missing. Also the pod-user-defaults(1) command can be used to create the default configuration file.

The `PoD.cfg` is a simple INI-like configuration file. Configuration file syntax is line based:

- A line in the form:

```
key_name=value
```

  gives a value to an option.

- A line in the form:

```
[section name]
```

  introduces a new section in the configuration file.

- The # character introduces a comment that spans until the end of the line.

  The option names are relative to the section names, so the following configuration file part:

```
[gui.accessibility]
visual_bell=yes
```

  is equivalent to

```
gui.accessibility.visual_bell=yes
```

Table 5.1. PoD server configuration

key	value	Description
server.work_dir	string (default: $HOME/.PoD)	PoD's working directory. Used by PoD modules to store temporary files, like pid files, for example. A string of the value will be evaluated

key	value	Description
		before it is used, it therefore can contain environment variables.
server.logfile_dir	string (default: $HOME/.PoD/log)	A path for PoD's log files. By the defined path PoD modules will place log files.
server.logfile_overwrite	yes/no (default: yes)	Defines whether PoD should overwrite its log files when starting a new session (PoD server's start/ restart cycle)
server.log_level	numeric (default: 1)	Defines the level of the log. There are following numeric values are allowed: • 0: Fault/Critical • 1: Fault/Critical/Info • 2: Fault/Critical/Info/Warning • 3: Fault/Critical/Info/Warning/ Debug
server.agent_shutdown_if_idle_for_sec	numeric (default: 1800)	Shut down a server if its idle time is higher than the defined value in seconds.
server.agent_local_client_port_(min/ max)	numeric (default: 20000/25000)	Recommended for advanced users only. The following range is used by PoD agent locally on the server host, when in the packet-forwarding mode. Each PROOF client gets its proxy redirected vie the ports from that range.
server.xproof_ports_range_(min/ max)	numeric (default: 21001/22000)	Recommended for advanced users only. PoD's automatic port mapping algorithms use this range to dynamically assign ports to xproof plug-in of xrootd when restarting a PoD server. In multi-user/ core environment, when there are many PoD processes on the same physical machine, the automatic port mapping prevents different PoD process of different users to disturb each other.
server.agent_ports_range_(min/ max)	numeric (default: 22001/23000)	Recommended for advanced users only. PoD's automatic port mapping algorithms use this range to dynamically assign ports to pod-agent when restarting a PoD server. In multi-user/core environment, when there are many PoD processes

key	value	Description
		on the same physical machine, the automatic port mapping prevents different PoD process of different users to disturb each other.
server.agent_threads	numeric (default: 5)	A number of threads in a thread pool. The thread pool is used by the pod-agent to distribute tasks of a proxy, when in the packet-forwarding mode.
server.agent_node_readbuffer	numeric (default: 5000)	A buffer size, used by the packet-forwarding algorithms (in bytes). It will be allocated for each PoD worker.
server.packet_forwarding	yes/no/auto (default: auto)	If workers are behind a firewall than PoD will use its packet-forwarding (PF) algorithms to maintain the PROOF traffic between server and workers. By setting this key to "yes" you force PoD to use PF in any case. If "auto" is set than PoD will decide on the fly whether to use PF for each worker individually based on the possibility to directly connect to worker.

Table 5.2. PoD worker configuration

key	value	Description
worker.work_dir	string (default: $POD_LOCATION)	PoD's working directory. Used by PoD modules to store temporary files, like pid files, for example. A string of the value will be evaluated before it is used, it therefore can contain environment variables.
worker.logfile_dir	string (default: $POD_LOCATION)	A path for PoD's log files. By the defined path PoD modules will place log files.
worker.logfile_overwrite	yes/no (default: yes)	Defines whether PoD should overwrite its log files when starting a new session (PoD worker's start/ restart cycle)
worker.log_level	numeric (default: 1)	Defines the level of the log. There are following numeric values are allowed: • 0: Fault/Critical • 1: Fault/Critical/Info • 2: Fault/Critical/Info/Warning • 3: Fault/Critical/Info/Warning/ Debug

key	value	Description
worker.set_my_rootsys	yes/no (default: yes)	Whether to use user's ROOTSYS on workers. If set to "yes", than the value of the worker.my_rootsys key, will be exported to the workers. See worker.my_rootsys for more details. If set to "no", PoD will download a default, pre-compiled version of ROOT according to WN's environment.
worker.my_rootsys	string (default: $ROOTSYS)	User's ROOTSYS to use on workers. If set_my_rootsys is set to "yes", than PoD will export bin and library locations of this ROOT version on the worker nodes. This is especially useful if you use shared home file system on the nodes where PoD workers are started or you know for sure the location of the ROOT installation on the worker nodes. A string of the value will be evaluated before it is used, it therefore can contain environment variables.
worker.agent_shutdown_if_idle_for_sec	numeric (default: 1800)	Shut down a worker if its idle time is higher than the defined value in seconds.
worker.xproof_ports_range_(min/max)	numeric (default: 21001/22000)	Recommended for advanced users only. PoD's automatic port mapping algorithms use this range to dynamically assign ports to xproof plug-in of xrootd when starting a PoD worker. In multi-user/core environment, when there are many PoD processes on the same physical machine, the automatic port mapping prevents different PoD process of different users to disturb each other.
worker.agent_node_readbuffer	numeric (default: 5000)	A buffer size, used by the packet-forwarding algorithms (in bytes). It will be allocated for each PoD worker.

Table 5.3. LSF plug-in configuration

key	value	Description
lsf_plugin.email_job_output	yes/no (default: no)	The parameter specifies whether job's output is sent to the user by mail. if "no" is set, output will be delivered to the log directory in std_[INDEX].err and std_[INDEX].out files

key	value	Description
lsf_plugin.upload_job_log	yes/no (default: no)	The parameter specifies whether to upload jobs log files from workers when PoD jobs are completed. Jobs log files include a full log of PROOF, XROOTD and pod-agent's log files.

Table 5.4. PBS plug-in configuration

key	value	Description
pbs_plugin.upload_job_log	yes/no (default: no)	The parameter specifies whether to upload jobs log files from workers when PoD jobs are completed. Jobs log files include a full log of PROOF, XROOTD and pod-agent's log files.
pbs_plugin.options_file	string (default: $POD_LOCATION/etc/Job.pbs.option)	This file can be used to provide addirtional PBS (qsub) options. Just create a file and set its path in pbs_plugin.options_file. Write valid qsub options in one line, like if you would write them in a command line when calling qsub. PoD will automatically use it (if exists) while submitting PBS jobs. Be advised, that the following options are reserved and are set by PoD, if you want to adjust them in anyway, then, please, contact PoD support and we will find a way. The reserved options are: -N, -q, -j, -V, -v.

Table 5.5. Grid Engine plug-in configuration

key	value	Description
ge_plugin.upload_job_log	yes/no (default: no)	The parameter specifies whether to upload jobs log files from workers when PoD jobs are completed. Jobs log files include a full log of PROOF, XROOTD and pod-agent's log files.
ge_plugin.options_file	string (default: $POD_LOCATION/etc/Job.ge.option)	PoD also supports an GE option file. If you want to provide some additional Grid Engine options to your PoD jobs submitted to OE cluster, to select some specific resource or something like that, than PoD gives you this possibility via an GE option file. Just create a file and set its path in ge_plugin.options_file. Write valid GE options in it. PoD will automatically use it (if exists) while

key	value	Description
		submitting GE jobs. See qsub man page of GE for more information on the option file (search for the "-@" option in the man page).

Table 5.6. Condor plug-in configuration

key	value	Description
condor_plugin.upload_job_log	yes/no (default: no)	The parameter specifies whether to upload jobs log files from workers when PoD jobs are completed. Jobs log files include a full log of PROOF, XROOTD and pod-agent's log files.
condor_plugin.options_file	string (default: $POD_LOCATION/etc/Job.condor.option)	PoD is shipped with a default Condor job description file, which is used to submit PoD jobs. If users need to use additional settings or requirements, in order to tune PoD job submission, these settings can be provided via a file specified by the condor_plugin.options_file option. Settings from this file will be added to the default PoD job description file. The options file should in the format of standard condor description files.

Important
All port ranges in the PoD configuration must not have intersections.

5.2. User's environment on workers

PoD provides a possibility for users to execute a custom environment script on workers before PoD processes start.

Users need to create a shell script file with the `$POD_LOCATION/etc/user_worker_env.sh` or `$HOME/.PoD/user_worker_env.sh` name and to code there all variables and commands to export to the workers. PoD will automatically transfer the script to each worker node and source it there.

For example, If I need to set the path to my ROOT installation on workers. I would create the following file.

```
#! /usr/bin/env bash

source /usr/local/pub/debian4.0/x86_64/gcc411-21/526-00/bin/thisroot.sh

export LD_LIBRARY_PATH=$MYLIBS/lib:$LD_LIBRARY_PATH
export MYVAR="some vallue :)"

# I need also my special profile there
source /etc/profile_extr
```

Be advised, that you need to recreate PoD worker package every time, when you modify the user script or if you removed it. To recreate the package, just call: **pod-prep-worker(1)**

The SSH plug-in has it's own machinery to setup custom environment on worker nodes. Please check the pod-ssh(1) documentation for more information.

5.3. XROOTD/XPROOFD

There is a default XROOTD configuration file, `$HOME/.PoD/etc/xpd.cf`. The file is generated from the template (`$POD_LOCATION/etc/xpd.cf.in`) each time PoD server is started. PoD uses this file to configure both local server and remote workers.

Tip

In XROOTD documentation you can find details of fine tuning of xrootd. But it is only recommended for advanced users.

The default xrootd configuration, which comes with PoD should be sufficient for basic operations. In most of use cases it is not needed to modify the configuration.

If you need additional xpd configuration settings, you can add custom xpd configuration files. PoD will scan for `$HOME/.PoD/user_xpd.cf*` and for `$POD_LOCATION/etc/user_xpd.cf*` and append the found files to the main xpd.cf. The star symbol in the file names can be change to any other symbol. For example, the following files will be appended to the main xpd.cf: `$POD_LOCATION/etc/user_xpd.cf0`, `$POD_LOCATION/etc/user_xpd.cf1`, `$POD_LOCATION/etc/user_xpd.cf2`.

PoD is only meant to help to setup a PROOF cluster on the fly using remote worker nodes. A data access is not a part of its responsibility.

6. Quick Start

PoD Quick Start:

1. Initialize PoD environment: Section 7.1, “Environment”
2. Start PoD server: Section 7.2, “Server”
3. Submit PoD workers to start dynamic PROOF cluster: Section 7.3, “Job Manager”
4. Check status of dynamic PROOF cluster: Section 7.4, “PROOF workers”
5. Use the PROOF cluster for an analysis: Section 7.6, “Analysis”
6. Restart PoD workers (if cluster needs to be reloaded): Section 7.3, “Job Manager”
7. Stop PoD server: Section 7.7, “How to shut down PoD”

The following is the example to illustrate the Quick Start. We use PoD with the SSH plug-in to setup our PROOF cluster on the bunch of the machines, which are described in the `pod_ssh.cfg` configuration file.

Detailed descriptions of the commands and of the configuration file can be found in the Chapter 10, *Command-line interface*.

PoD Environment:

```
cd [PoD INSTALL DIRECTORY]
source PoD_env.sh
```

Important

The current implementation of the SSH plug-in requires users to have a public key access (or password less) to destination remote hosts (worker nodes).

Starting the cluster:

```
pod-server start
pod-server status
pod-ssh -c pod_ssh.cfg submit
pod-ssh status
pod-info -n
pod-info -l
```

The Dynamic PROOF cluster is ready to perform user's analysis code...

Remote Environment

With SSH plug-in it is very often the case, that PoD can't start workers, because xproofd/ROOT is not in the PATH on worker nodes. If your PoD job fails, just after submission it shows DONE status. You may want to check the remote log files see the section called “Examples” from the worker nodes and if it says that there are problems to start xproofd, then you need to customize environment on WNs. This could happen since with a batch SSH login in some systems you don't get your /etc/profile script called (login script) and there is no environment variables, like for normal login users.

To solve this issue, users either can specify the full path to desired ROOT version on the worker nodes in the `PoD.cfg` or just use Section 5.2, “User's environment on workers”. The last one is very much advisable.

If needed we can restart it:

```
pod-ssh clean
pod-ssh submit
pod-ssh status
pod-info -n
```

And finally, lets shut down out PoD(PROOF) cluster:

```
pod-server stop
pod-ssh clean
pod-ssh status
```

7. How to run

7.1. Environment

In order to enable PoD's environment you need to source the `PoD_env.sh` script. The script is located in the directory where you installed PoD.

```
cd [POD INSTALLATION]
source PoD_env.sh
```

Also don't forget, before starting PoD the ROOT should be in the PATH as well.

7.2. Server

local and remote PoD servers

There are two ways you can use PoD: as a local server and a remote one. On how to use a remote PoD server please check the pod-remote(1) command.

Further in this chapter are the instructions on how to use a local PoD server.

Use the pod-server command to start/stop/status PoD servers.

```
pod-server start
```

7.3. Job Manager

The next step is to submit remote PoD workers using PoD's job manager. These PoD workers will automatically setup your PROOF workers on remote hosts. Starting from version 2.0.7 the PoD project supports plug-ins. To submit remote jobs job manager plug-ins are used. That means PoD could be used on different resources like Grid, Cloud, RMS or just simple machines with only an ssh access on them. It also possible to use a combination of plug-ins to get PROOF workers on Grid worker nodes and local batch machines in the same time.

In order to setup a dynamic PROOF cluster on RMS such as gLite, LSF, PBS, Condor, LoadLeveler or (Oracle/Sun)GridEngine, use the pod-submit command.

The following simple example illustrates a submission of 15 workers to an LSF farm using the "proof" queue.

```
pod-submit -r lsf -q proof -n 15
```

Use PoD user defaults to tune individual settings of plug-ins: gLite, LSF, PBS, Condor, LoadLeveler or (Oracle/Sun)GridEngine.

If there is no RMS available, you can use PoD's SSH plug-in. Please see Chapter 8, *SSH plug-in* for more details.

7.4. PROOF workers

As soon as a single job reaches remote worker node (WN), it tries to connect to PoD server to transfer information about itself and environment of WN. When negotiations are done and PoD server accepts WNs, it became a normal PROOF Worker for the user.

It is not required to wait until all requested workers will be connected. Users could start analysis after reasonable number of workers are on-line, even after the first connected worker one could start the analysis. When other workers arrive, the ROOT (PROOF) session must be restarted in order to reconnect to the newly arrived workers.

Tip

PoD supports reconnection. That means if your analysis has a bug or a root session crashed you don't need to resubmit PoD jobs. You just need to close current root session, open it again. PoD will manage reconnection with its worker nodes automatically. Worker nodes will be on-line until the pod-agent service is on-line or until s Grid and/or batch queue time is over.

Use the pod-info command to find out a number

```
pod-info -n
```

or to list

```
pod-info -l
```

available PROOF workers.

The **pod-info -l** command can be also used to check, whether a direct connection (preferable) or a packet forwarded connection is used to connect PROOF server to workers. PoD y default automatically chooses the type of connections.

If PoD server can't directly connect to its workers on xpd port, then the packet forwarded connection is used. With this type of connection, some PROOF functions will be limited. For example, workers can't be used as parallel sub-mergers, since a direct connection between workers will be required. We therefore recommend to open an xpd port range (PoD user defaults: worker.xproof_ports_range_(min/max)) for incoming connections on the worker nodes. This will help PoD to set up the most efficient type of connection.

7.5. PROOF Connection String

PROOF connection string - is an URL which is used as a parameter to the TProof::Open method. This URL actually contains an address of PROOF master, its host and port.

Every time PoD is restarted it uses its automatic port mapping machinery to assign TCP ports to xproofd and other daemons. That means, a PROOF master port can always be a different one. In order always get the actual port and even the whole PROOF connection string the pod-info(1) can be used.

For an example analysis, please see Chapter 9, *How to test*.

7.6. Analysis

Now when your remote PROOF workers (PoD workers) are on-line, you can process you ROOT/PROOF analysis normally, if it would be a usual PROOF session.

For an example analysis, please see Chapter 9, *How to test*.

7.7. How to shut down PoD

In order to shut down PoD, a PoD server should be stopped.

```
pod-server stop
```

7.8. if something is wrong

If something goes wrong, something doesn't work as expected, please, check the log files first.

Table 7.1. PoD log files

Name	Location	Description
pod-agent.server.log	server.logfile_dir/pod-agent.server.log	This file contains a log messages of the pod-agent, which runs on the user interface.
xpd.log	server.logfile_dir/PoDServer	This is an XROOTD log file.

All job manager plug-ins are also able to deliver the logs from worker nodes. Please refer to the plug-ins configuration for more details.

If you still can't resolve the issue or have something to report, use Chapter 13, *Support*.

8. SSH plug-in

8.1. CLI

For users convenience PoD provides a command line interface. One can also use PoD CLI to submit PoD jobs, instead of using GUI. Meet the CLI documentation and check out the pod-ssh(1) reference for further information.

Before you start, check that PoD Section 7.2, “Server” is running.

The following simple example illustrates a submission of a number of PoD workers (described in the `pod_ssh.cfg` configuration file) to a bunch of the machines via SSH.

```
pod-ssh -c pod_ssh.cfg submit
```

check the status of PoD workers:

```
pod-ssh status
```

There are could be the following values of the status:

- RUN - PoD jobs is running,
- DONE - PoD job is done, means PoD worker is not running on that worker node. It could be also the case that worker failed to start,
- CLEAN - PoD worker has been cleaned,
- UNKNOWN - it is not possible to retrieve the status of that worker.

Remote Environment

With SSH plug-in it is very often the case, that PoD can't start workers, because xproofd/ROOT is not in the PATH on worker nodes. This could happen since with a batch SSH login in some systems you don't get your /etc/profile script called (login script) and there is no environment variables, like for normal login users. If your PoD job fails, just after submission it shows DONE status. You may want to check the remote log files see the section called “Examples” from the worker nodes and if it says that there are problems to start xproofd, then you need to customize environment on WNs. To solve this issue use inline BASH script.

Now check the status of your dynamic PROOF clusters. The following commands show a number/list of available PROOF workers, which have been already set up and are online:

```
pod-info -n
pod-info -l
```

and finally clean the PoD cluster. The cleaning needs to be performed when user is done with his/her dynamic PROOF cluster or want to refresh workers (in this case, you need to submit workers again, after the cleaning). BTW, no need to stop pod-server.

```
pod-ssh clean
pod-ssh status
```

Important

The cleaning of the workers is very important in order to keep the remote environment safe and clean. Also the cleaning procedure can deliver log files from the workers, see the section called "Examples". Unfortunately SSH plug-in can't automatically decide when to clean the workers, you therefor is responsible to do it.

At the end, check that you shut you PoD server down - Section 7.7, "How to shut down PoD".

Detailed descriptions of the command and of the configuration file can be found in the pod-ssh(1) reference manual.

8.2. Configuration

9. How to test

9.1. Simple test

The simplest way to test your dynamic PROOF cluster is just to run some simple analysis on it or use the new benchmark framework: TProofBench.

Now if you want to perform TProofBench tests. Use the Chapter 7, *How to run* to setup your dynamic PROOF cluster. As soon as you get required number of workers you can execute the following commands:

```
$ root
root[] TProofBench pb(gSystem->GetFromPipe("pod-info -c"))
root[] pb.RunCPU()
```

or for ROOT v5.30 (or higher):

```
$ root
root[] TProofBench pb("pod://")
root[] pb.RunCPU()
```

10. Command-line interface

Name

pod-server — Manages PoD server

Synopsis

`pod-server` {[start] | [restart] | [stop] | [status] | [status_with_code] | [getbins]}

Description

Using this command users can start/stop/restart PoD server and force to download pre-compiled PoD WN binaries from the central PoD repository. PoD server currently works with two daemons, namely xproofd and pod-agent.

When the `status` argument is used, **pod-server** will show running processes including their process IDs and used TCP ports. For user convenience the **pod-server** command with the `status` option prints also a PROOF connection string (master_host:xproofd_port), which can be used as an argument to `TProof::Open` in PROOF analysis scripts. However it is recommended to use pod-info(1) in order to retrieve the current connection string.

Options

`start`
: Start PoD server.

`restart`
: Restart PoD server.

`stop`
: Stop PoD server.

`status`
: Request the status information. It will show which processes are running, under which PIDs and which TCP ports are used.

`status_with_code`
: This option is exactly the same as `status`. The only difference is that when the option is used the **pod-server** utility exits 0 if PoD server is running, and >0 if it doesn't.

`getbins`
: Force PoD server to download workers pre-compiled binaries from the PoD repository. The binaries than saved to `$POD_LOCATION/bin/wn_bins` directory.

Name

pod-info — Shows information about PoD and PROOF workers.

Synopsis

`pod-info` *(1) general options (2) information options (3) connection options*
(1) `[-h, --help][-v, --version][-d, --debug][-b, --batch]`
(2) `[[-c, --connection_string]|[-l, --list]|[-n, --number]|[-s, --status]|[--xpdPid]|[--xpdPort]|[--agentPid]|[--agentPort]]`
(3) `[--remote` *`arg`*`][--remote_path` *`arg`*`][--ssh_opt` *`arg`*`][--ssh_open_domain` *`arg`*`]`

Description

One can use **pod-info** to retrieve different kinds of information about and from PoD. For example **pod-info** could help to find out whether PoD server running or not, how many PROOF workers are already online and which exactly. Please see the section called "Options" to find out all kinds of information **pod-info** can retrieve and show.

By default **pod-info** tries to find and connect to a local PoD server. A PoD server considered to be a local one if the **pod-info** and the PoD server run under the same user id. It could be the same machine or different machines but with a shared home file system. If none of local PoD servers are detected, **pod-info** will check for any PoD server managed by the pod-remote(1) command and will connect to the server if found.

When you want to retrieve information about remote PoD servers, than you need to use the `--remote` option. Using this option you can specify an ssh connection string, where a remote PoD server is running. The **pod-info** command will first try to find the running PoD server on that host and than process user requests on that server. In the section called "Examples" you will find some use cases.

The **pod-info** utility exits 0 on success, and >0 if an error occurs.

Options

`-h, --help`
: Produce help message.

`-v, --version`
: Version information.

`-d, --debug`
: Show debug messages. This option enables a debug mode and helps in some cases to understand what is going wrong.

`-b, --batch`
: Enable the batch mode. For example, in case when the `--remote` option also used, than there will be no password prompts or any interaction with a user. The utility will try to use SSH public key authentication and will fail if it's not working.

  The batch mode is very useful, when **pod-info** is used in a ROOT/PROOF script to retrieve a connection string (see Example 10.3, "PROOF connection string"). In this case you want **pod-info** to return either a PROOF connection string or an empty string in case of an error and no prompts of any kind.

`-c, --connection_string`
: Show a PROOF connection string, which could be passed to the TProof::Open method as an argument (see Example 10.3, "PROOF connection string").

`-l, --list`
: List all available PROOF workers.

`-n, --number`
: Report a number of currently available PROOF workers.

`-s, --status`
Show PoD server status.

`--remote` *`arg`*
A connection string in form of user@host.fqdn. Directs **pod-info** to use SSH to detect and connect to a remote PoD server.

`--remote_path` *`arg`*
A working directory of the remote PoD server. It is very important either to write an explicit path or use quotes, so that shell will not substitute local variable in the remote path. (default: ~/.PoD/)

`--ssh_opt` *`arg`*
If needed, users can provide additional SSH options, which will be used by **pod-info** in all SSH communications.

`--ssh_open_domain` *`arg`*

Note

The `--ssh_open_domain` is in development. So far **pod-info** can only work with remote PoD servers which are at least directly accessible via SSH.

The name of a third party machine open to the outside world and from which direct connections to the server are possible. The optional argument, can be used when PoD server machine is not directly accessible from outside via SSH.

Examples

Example 10.1. PoD version information

```
pod-info -v
```

Get PoD version from a remote PoD server on the machine *`server.fqdn`* (it could be also on a shared home file system there):

```
pod-info --remote user@server.fqdn -v
```

Example 10.2. available PROOF workers

a number of workers:

```
pod-info -n
```

the same from the remote PoD server:

```
pod-info --remote user@server.fqdn -n
```

a list of workers:

```
pod-info -l
```

or all together:

```
pod-info -nl
```

Example 10.3. PROOF connection string

```
pod-info -c
```

the same from the remote PoD server:

```
pod-info --remote user@server.fqdn -c
```

or if the remote server has a non-default working folder:

```
pod-info --remote user@server.fqdn --remote_path "~/pod/work_dir/" -c
```

use this command in a ROOT script or your analysis code directly:

```
TProof::Open(gSystem->GetFromPipe("pod-info -c"));
```

with a remote PoD server:

```
TProof::Open(gSystem->GetFromPipe("pod-info --remote user@server.fqdn -c -b"));
```

note, that we use the `-b` option, because we can't have any prompt in out ROOT script and therefore the call to **pod-info** must be silent. Since it will return an empty string in case of failure, users may want to check:

```
std::string url(gSystem->GetFromPipe("pod-info --remote user@server.fqdn -c -b"));
if( url.empty() )
{
   // PoD server is not running
   // print out an error message...
   return 1;
}
TProof::Open( url.c_str() );
```

Example 10.4. PoD server status

```
pod-info -s
```

the same from the remote PoD server:

```
pod-info --remote user@server.fqdn -s
```

Name

pod-user-defaults — Retrieves values from PoD user defaults configuration file.

Synopsis

`pod-user-defaults` [`-h, --help`] [`-v, --version`] [`-c` *`file`*`, --config=`*`file`*] [`--key=`*`name`*]

Description

The **pod-user-defaults** command can be used to retrieve values from the PoD user defaults configuration file for any given keys. The PoD user defaults configuration "`PoD.cfg`" is a general PoD settings file, where user can tune PoD for a specific environment. The file usually can be found in `$HOME/.PoD/etc` or in `$POD_LOCATION/etc`.

Options

`-h, --help`
: Show summary of options.

`-v, --version`
: Version information.

`-c` *`file`*`, --config=`*`file`*
: PoD user-defaults configuration file.

`--key=`*`name`*
: The **pod-user-defaults** retrieves a value for the given key. A key must be specified with a its section name (separated by a dot), for example to find out a working directory on the PoD server, request the value for the following key: "server.wrk_dir".

Name

pod-prep-worker — Prepares a worker package - all elements of PoD which need to be uploaded to a worker node.

Synopsis

```
pod-prep-worker
```

Description

The **pod-prep-worker** command creates a tar-zipped archive of all files which are required on PoD worker. The command prepares a PoD worker package. This package and the **PoDWorker.sh** job script are all what is required to start a PoD worker.

The **pod-prep-worker** command must be issued on the PoD server machine and only when PoD server is up an running. Otherwise the package will be not created.

Name

pod-submit — Submits PoD jobs using a defined resource management system.

Synopsis

`pod-submit [-h] [-l] [-r condor|ge|loadleveler|lsf|pbs|glite] [-q queue] [-n X]`

Description

Use this command to manually submit PoD workers to a defined resource management system. The command currently supports:

- LSF (Load Sharing Facility),
- PBS Pro/OpenPBS/Torque (Portable Batch System),
- Grid Engine (Oracle/Sun Grid Engine),
- Condor,
- LoadLeveler (IBM Tivoli Workload Scheduler LoadLeveler),
- gLite.

Use **pod-submit `-l`** to find out the list of available and supported CLI plug-ins.

Note

The pod-submit can't be used to submit SSH jobs. In order to use PoD SSH plug-in, please check the pod-ssh(1) reference manual.

The `$POD_RMS_DEFAULT_QUEUE` environment variable can be used to define a default RMS queue. If no "-q" option is provided to "pod-submit", then the value of this variable is used.

The **pod-submit** utility exits 0 on success, and >0 if an error occurs.

Options

`-h`
: Show summary of options and exit.

`-l`
: Show all available RMS plug-ins.

`-r condor|ge|loadleveler|lsf|pbs|glite`
: A name of the resource management system to use.

`-q queue`
: Submit the jobs to specified `queue`. (default: "proof" if the value of `$POD_RMS_DEFAULT_QUEUE` is empty)

  In case of gLite plug-in, queue parametr must define a CREAM CE including a desired queue, for example:

```
pod-submit -r glite -q atlasce2.lnf.infn.it:8443/cream-pbs-atlas_short -n 30
```

`-n X`
: Specify a desired number or PROOF workers, where the `X` option defines the number of workers. (default: 10)

Name

pod-ssh — Submits, retrieves statuses and cleans PoD workers using SSH connections.

Synopsis

`pod-ssh[-h, --help][-v, --version][-d, --debug][-c` *`file`*`, --config=`*`file`*`][-e` *`arg`*`, --exec=`*`arg`*`][-t` *`arg`*`, --threads=`*`arg`*`][--logs][--for-worker` *`arg`*] {[submit] | [clean] | [fast-clean] | [status]}

Description

The **pod-ssh** command can be used to submit and clean PoD workers using an ssh connection.

Important

The current implementation requires users to have a public key access (password less) to destination remote hosts (worker nodes).

The **pod-ssh** command takes PoD's ssh plug-in configuration file as input. The configuration file is a comma-separated values (CSV) file. Fields are normally separated by commas. If you want to put a comma in a field, you need to put quotes around it. Also 3 escape sequences are supported.

Table 10.1. PoD's ssh plug-in configuration fields

1	2	3	4	5
id (must be any unique string). This id string is used just to distinguish different PoD workers in SSH plug-in.	a host name with or without a login, in a form: login@host.fqdn	additional SSH params (could be empty)	a remote working directory	a desired number of PROOF workers (could be empty). If this parameter is empty, than PoD will spawn as many PROOF workers on that host as CPU cores.

An example of a configuration file:

```
r1, anar@lxg0527.gsi.de, -p24, /tmp/test, 4
# this is a comment
r2, user@lxi001.gsi.de,,/home/user/pod,16
125, user2@host, , /tmp/test,
```

The **pod-ssh** command remembers last entered config-file pathname and next time you want to use **pod-ssh** with the same config file, you can just call **pod-ssh** without providing the `--config` option. The command will always use the latest given setting. In order to use feature BOOST 1.41.0 (or higher) is required.

Environment on Worker Nodes

With SSH plug-in it is very often the case, that PoD can't start workers, because xproofd/ROOT is not in the PATH on worker nodes. This could happen since with a batch SSH login in some systems you don't get your /etc/profile script called (login script) and there is no environment variables, like for normal login users. If your PoD

job fails, just after submission it shows DONE status. You may want to check the remote log files see the section called “Examples” from the worker nodes and if it says that there are problems to start xproofd, then you need to customize environment on WNs. To solve this issue, users either can specify the full path to desired ROOT version on the worker nodes in the `PoD.cfg`, in case when all WNs have the same version pf ROOT located by the same path. But more advisable solution is to use inline bash script.

Inline BASH script

User can define remote environment for PoD SSH worker nodes via a so called inline BASH script. To define a script just use @bash_begin@ and @bash_end@ tags in your PoD SSH configuration file. For example:

```
@bash_begin@
    # GSI
    . /etc/profile.d/gsi.sh
    . rootlogin 527-06b-xrd
@bash_end@

r1, anar@lxg0527.gsi.de, -p24, /tmp/test, 4
# this is a comment
r2, user@lxi001.gsi.de,,/home/user/pod,16
125, user2@host, , /tmp/test,
```

Everything what PoD find between those tags will be considered as an environment script and will be sourced on each worker node listed in that configuration file.

Bu using this feature, users are able to define different configuration files for different clusters, each of which can define its own list of worker nodes and an environment script accordingly.

Be advised, if inline BASH script is found, then PoD will not use user_worker_env.sh

The **pod-ssh** utility exits 0 on success, and >0 if an error occurs.

Options

`-h, --help`
: Show summary of options.

`-v, --version`
: Version information.

`-d, --debug`
: Show debug messages. This option enables a debug mode and helps in some cases to understand what is going wrong.

`-c` *`file`*`, --config=`*`file`*
: PoD's ssh plug-in configuration file. A workers description file.

`-e` *`arg`*`, --exec=`*`arg`*
: Execute a local shell script on remote worker nodes

`-t` *`arg`*`, --threads=`*`arg`*
: It defines a number of threads in pod-ssh's thread pool. Min value is 1, max value is (Core*2). Default: 5

`--logs`
: Download all log files from the worker nodes. Can be used only together with the `--clean` option. This command delivers all log files from the worker nodes. Logs are copied to PoD log directory, the path to which is configurable via PoD user defaults.

`--for-worker` *`arg`*
: Perform an action on defined worker nodes. (arg is a space separated list of WN names) Can only be used in connection with "submit", "clean", "fast-clean", "exec".

`submit`
: Submit PoD workers according to the entries in the configuration file.

`clean`
: Clean all PoD workers according to the entries in the configuration file.

`fast-clean`
: The fast version of the clean procedure. It only shuts worker nodes down. It doesn't actually clean workers' directories.

`status`
: Request status of PoD workers listed in the configuration file.

  There are could be the following values of the status:

  - RUN - PoD job is running,
  - DONE - PoD job is done, means PoD worker is not running on that worker node. It could be also the case that worker failed to start,
  - CLEAN - PoD worker has been cleaned,
  - UNKNOWN - it is not possible to retrieve the status of that worker.

Examples

Example 10.5. Submit PoD jobs via SSH

```
pod-ssh -c pod_ssh_config_file submit
```

Example 10.6. Check the status of PoD jobs submitted via SSH

```
pod-ssh status
```

Check the amount of available PROOF workers:

```
pod-info -n
```

or

```
pod-info -l
```

Example 10.7. Clean PoD jobs submitted via SSH

```
pod-ssh clean
```

also you can clean and download all log files from the WNs

```
pod-ssh clean --logs
```

Example 10.8. Clean only specific worker nodes

```
pod-ssh --for-worker r1 r2  clean
```

Name

pod-remote — Using this command users can start/stop/restart remote PoD servers. The utility can also be used to execute arbitrary commands on remote PoD servers, such as PoD job submissions.

Synopsis

`pod-remote` *(1) general options (2) connection options (3) commands*
(1) `[-h, --help][-v, --version][-d, --debug][-c file, --config=file]`
(2) `[--remote arg][--ssh-opt arg][--ssh-open-domain arg][--env-local arg][--env-remote arg]`
(3) `[[--start]|[--stop]|[--restart]|[--command cmd]]`

Description

In order to use **pod-remote** BOOST 1.41.0 (or higher) is required.

Important

The current implementation requires users to have a public key access (password less) to destination remote hosts (PoD servers).

The **pod-remote** command offers a possibility to fully control remote PoD servers. A PROOF cluster created using **pod-remote** is accessed via SSH tunnels, which are automatically managed by **pod-remote**.

Using **pod-remote** it is possible to start/restart/stop remote PoD servers. It is also possible to submit PoD jobs from remote PoD servers in order to set remote PROOF clusters up.

Most importantly, **pod-remote** automatically creates and handles SSH tunnels for remote PoD servers, so that these servers can be used only via SSH connection - outside of a firewall. Tunnels stay alive until remote server is alive or you restart/stop pod-remote. The pod-remote command creates a background daemon, which regularly checks the status of the remote PoD server and manages tunnels. Another important feature of **pod-remote** is its integration into PoD, see the section called “Examples”.

The **pod-remote** command remembers all connection options values and next time you want to use **pod-remote** with the same server, you can omit these arguments and just call: **pod-remote `--start/stop/restart`** without `--remote` and `--env`. The command will always use the latest given settings. If you want to change the server, just provide new arguments values.

The **pod-remote** utility exits 0 on success, and >0 if an error occurs.

Options

`-h, --help`
Produce help message.

`-v, --version`
Version information.

`-d, --debug`
Show debug messages. This option enables a debug mode and helps in some cases to understand what is going wrong.

`-c file, --config=file`
Specify an options file with the pod-remote command line options.

`--remote arg`
A connection string including a remote PoD location. For example: loginname@serverhostname:/PoD/location/pathname

`--ssh-opt` *arg*
: If needed, users can provide additional SSH options, which will be used by **pod-remote** in all SSH communications.

`--ssh-open-domain` *arg*
: The name of a third party machine open to the outside world and from which direct connections to the server are possible. The optional argument, can be used when PoD server machine is not directly accessible from outside via SSH.

`--env-local` *arg*
: A full path to a local environment script, which will be executed on the remote-end before PoD starts the server. This is needed in order to get working environment on the remote host before PoD server can be started. In most of the case you would need to source proper ROOT environment. You can, of course, also set some other env. variables in the script, if needed.

`--env-remote` *arg*
: The same as `--env-local`, but the script is located on the remote machine and will be sourced there.

`--start`
: Start remote PoD server.

`--stop`
: Stop remote PoD server.

`--restart`
: Restart remote PoD server.

`--command`
: Execute arbitrary commands.

Examples

Example 10.9. Using remote PoD server

Let's say, I have two machines A and B.

```
A - is my laptop, for example.
B - is a machine (host: lxg0527) standing somewhere at my work place.
   From this machine I can submit jobs to my batch system (LSF, for example)
   or use it as a server for PoD SSH plug-in.
```

On both machines I have PoD installed.

All the following commands I issue from my laptop (machine A).

Start the remote PoD server:

```
pod-remote --start \
    --remote manafov@lxg0527.gsi.de:/home/manafov/3.6/ \
    --env-local ../GSI_env_5_27.sh
```

The command above starts a remote PoD server on the host `lxg0527.gsi.de` under the user `manafov` and uses PoD installed in the `/misc/manafov/PoD/3.5.75.gbecd4` directory. To initialize the proper environment on the remote host the `../GSI_env_5_27.sh` script is used.

If everything is OK and remote server is started, **pod-remote** will create and manage special SSH tunnels from machine A to B. So, the whole PoD communication and PROOF requests will go via these tunnels.

To set our remote PROOF cluster up, now we need to submit PoD jobs from the remote Server (in case of RMS plug-ins):

```
pod-remote --command "pod-submit -r lsf -n 50 -q my_lsf_queue"
```

or in case of the SSH plug-in

```
pod-remote --command "pod-ssh -c pod_ssh.cfg submit"
```

Using `--command`, you can execute any command vie SSH on the remote server.

Now, you can just use pod-info(1) as usual, as if everything would run locally:

```
pod-info -s
pod-info -c
pod-info -n
...
```

The **pod-info** automatically detects that there is a pod-remote-managed server and will gather the information directly from it via the SSH tunnels. It means, of course, that to connect from your local machine to your remote PoD/PROOF cluster you need just to use:

```
TProof::Open(gSystem->GetFromPipe("pod-info -c"));
```

To stop the remote PoD server use:

```
pod-remote --stop
```

To start the same server again use the following command. Note missing `--remote`. You don't need it. The command remembers your last valid settings.

```
pod-remote --start
```

To restart the server:

```
pod-remote --restart
```

11. Tips

11.1. Handling large outputs via ROOT files

There is a great feature in PROOF to help merging big output objects. To enable this feature in your dynamic PROOF cluster managed by PoD, you need to use an automatic rootd transport instead of the default one (xrootd), because by default PoD doesn't use xrootd. The rootd transport was implemented in ROOT 5.28 (see the patch).

In order to switch on the rootd transport you need to add "**xpd.rootd allow**" to your custom XPD configuration file.

12. Known Issues

12.1. General Issues

/tmp on worker nodes

The `/tmp` directory on remote workers must be open for r/w. PROOF and ROOT writes there. I have redirected all possible temporary files to PoD working directory, but there are still some files, which ROOT/PROOF writes to the /tmp, it includes sockets files of proof/xrootd.

PoD on AFS

Since AFS doesn't support pipes you need to change the PoD server working directory in PoD user defaults configuration, so that a new directory will not reside on AFS anymore. Something like that should work:

```
[server]
#
# PoD working directory
#
work_dir=/tmp/manafov/
```

WARNING: File /afs/.../pod-worker is not readable by condor

See the section called “Condor and AFS”

It seems I run always X slaves, but I requested Y.

PoD setups workers on the remote nodes and it makes PROOF master to think (only when PoD packet-forwarding connection is used), that all of his workers are on the localhost. Actually PoD hides remote PROOF workers from the PROOF server and acts as a "proxy" between them. And since default value for `PROOF_MaxSlavesPerNode` is 2, therefore only 2 slaves get packages. Since all slaves (for PROOF server) are on the localhost, the other Y-2 workers won't get packages.

See for more information PROOF Wiki:

```
PROOF_MaxSlavesPerNode
Type: int
Description: Parameter for the packetizers. Limit the number of slaves accessing data o
Default Value:
In TPacketizer the default value is 4.
In TPacketizerAdaptive and TPacketizerProgressive it is 2.
```

Note

From other source of information, it looks like the default number of workers reading remotely from one file node (worker machine) is not "2", but a number of CPU cores of the master node.

In order to resolve this issue, you need to change one variable of your PROOF session (50 is only an example):

```
```

```
proof->SetParameter( "PROOF_MaxSlavesPerNode", (Long_t)100 );
```

Hopefully in the future, this will be possible to do through XROOTD configuration file and PoD will manage it for you automatically.

gLite environment issue at CERN's LSF

If PoD doens't work for you out of the box at CERN on LSF, PoD jobs fail to start PoD workers, then most probably you are facing so called "gLite environment" issue. Check you the logs from PoD jobs and if you see something like this in you std_XXX.out of the job, then most probably xproofd will fail to start:

```
LD_LIBRARY_PATH=/tmp/PoD_cgmwU22625:/opt/d-cache/dcap/lib:/opt/d-cache/dcap/lib64:/opt/
/opt/lcg/lib:/opt/lcg/lib64:/usr/lib64:/afs/cern.ch/user/m/mbellomo/PoD/3.6/lib:/afs/ce
/afs/cern.ch/sw/lcg/external/Boost/1.47.0_python2.6/x86_64-slc5-gcc43-opt//lib:/afs/cer
/afs/cern.ch/sw/lcg/contrib/gcc/4.3.5/x86_64-slc5-gcc34-opt/lib64:/afs/cern.ch/sw/lcg/c
/afs/cern.ch/sw/lcg/contrib/gmp/4.2.2/x86_64-slc5-gcc34-opt/lib:/opt/classads/lib64/:/o
```

Note "glite" in the paths.

Just as a workaround, you can try to use Section 5.2, “User's environment on workers” in this file you need just two lines

```
#! /usr/bin/env bash
export LD_LIBRARY_PATH=/afs/cern.ch/sw/lcg/external/qt/4.4.2/x86_64-slc5-gcc43-opt/lib:
/afs/cern.ch/sw/lcg/external/Boost/1.44.0_python2.6/x86_64-slc5-gcc43-opt//lib:/afs/cer
/afs/cern.ch/sw/lcg/contrib/gcc/4.3.5/x86_64-slc5-gcc34-opt/lib64:/afs/cern.ch/sw/lcg/c
/afs/cern.ch/sw/lcg/contrib/gmp/4.2.2/x86_64-slc5-gcc34-opt/lib:/afs/cern.ch/alice/libr
/afs/cern.ch/alice/library/afs_volumes/vol12/AliRoot/lib/tgt_linuxx8664gcc
```

or any sutable LD_LIBRARY_PATH you like, just without gLite libs.

Another solution is to check what inserts glite environemnt to your LSF jobs and get rid of it.

12.2. Condor Issues

Condor and AFS

If your home is on AFS, than you need to give permissions to Condor to access some of your PoD directories. Namely, Condor needs to have full access to the following folders: `$HOME/.PoD/wrk` and `$HOME/.PoD/log`. The last path is the default path for PoD logs. If you changed this path in PoD user defaults settings and a new log directory is also on AFS, than you need to open it accordingly.

You can do that by issuing the following commands:

```
fs setacl -dir $HOME/.PoD/wrk -acl system:anyuser rlidwk
fs setacl -dir $HOME/.PoD/log -acl system:anyuser rlidwk
```

12.3. Grid Issues

ClassAds and Namespace

One may want to compile CLASSADS with namespace support, because gLite UI contains CLASSADS which compiled without support of namespaces, though some of gLite API libraries (WMSUI for example) require classads with namespace support. This issue will prevent GAW to be build properly.

Download classads-0.9.9 from here.

```
tar -xzvf classads-0.9.9.tar.gz
cd classads-0.9.9
./configure --enable-namespace
make
make install
```

Be advised that in some Linux distributions there is the ClassAds package. For example Fedora 9:

```
yum install classads
yum install classads-devel
```

GLOBUS Libs Relocation

If you have gLiteUI installed from relocatable tarball, then you may face this gLite bug by having the following (or similar ones) error messages while compiling GAW library:

```
grep: /opt/globus/lib/libglobus_ftp_control_gcc32dbg.la: No such file or directory
/bin/sed: can't read /opt/globus/lib/libglobus_ftp_control_gcc32dbg.la: No such file or
libtool: link: `/opt/globus/lib/libglobus_ftp_control_gcc32dbg.la' is not a valid libto
```

One of the solutions would be to just copy Globus libs to /opt/:

```
cd $GLOBUS_LOCATION
mkdir -p /opt/globus
cp -rv lib /opt/globus/
```

GridSite headers missing

If you have gLiteUI installed from relocatable tarball, then you may face this gLite bug. This issue will prevent GAW to be build properly.

One of the solutions would be to just get these headers from somewhere.

globus_config.h is missing

Since long time gLite UI_TAR installation (I suspect gLite UI as well) is missing "globus_config.h" (see CERN Savannah - gLite Bug #31180). gLite API is referencing to this file, but is not providing it. Users therefore should find it some where, to let GAW to use gLite API.

13. Support

The PoD Support home page is your portal to help and support for the product.