
Grid Documentation Documentation

Release 1.0

Grid Support <helpdesk@surfsara.nl>

March 11, 2016

1	General	3
2	Basics	13
3	Advanced topics	29
4	Local jobs on the Life Science Grid	71
5	Best practices	79
6	Service implementation	111
7	Tutorials	121
8	Frequently asked questions	129
9	Indices and tables	145

Welcome to the documentation for using the Grid services at [SURFsara](#). The information in this tutorial will help you get started with the Grid, learn best techniques to successfully port your application to the Grid infrastructure and stay up-to-date with our system developments. We welcome your comments at helpdesk@surfsara.nl or your *contribution* to help us improve the documentation.

Need help?

Do you need help with this tutorial? We are more than willing to assist! Just contact us at helpdesk@surfsara.nl.

1.1 About the Grid

In this page you will find general information about the Grid, how it works and what is suited for:

Contents

- *About the Grid*
 - *Introduction to Grid*
 - *How it works*
 - *To use the Grid or not*

1.1.1 Introduction to Grid

The Grid is a loosely coupled collection of heterogeneous resources with a large variety in purpose and reliability. The Grid consists of multiple geographically distributed compute clusters interconnected with fast network. Grid computing is different from conventional high performance computing such as cluster computing in that Grid systems typically have each processor core perform an independent task.

More about Cluster computing basics?

See also:

Check out our mooc video [Cluster Computing](#)

In addition, Grid clusters are not exclusively connected to their own storage facilities but can be accessed transparently from all Grid compute clusters, and as such form a virtual supercluster. Grid systems and their applications are more scalable than classic clusters.

The Grid is especially suited, therefore, to applications that cannot be solved in a single cluster within a reasonable timeframe. Common examples are the Large Hadron Collider (LHC) experiments, large-scale DNA analyses and Monte-Carlo simulations.

1.1.2 How it works

As a user you connect to the Grid by connecting to a so-called **User Interface** (UI) system via secure shell. Once you have received the right credentials for a UI (User Interface) (see [Preparation](#)) you are set to go.

In general your task needs to be split into smaller units, called **jobs**, that each fit a certain set of boundary conditions in terms of resources (typically runtime, memory, disk size). For the jobs to be executed on the Grid, a job slot needs to be selected based on the boundary conditions that suit the requirements for these jobs. The way to do this is to describe each job in terms of a Job Description Language (JDL), where you list which program should be executed and the requirements of the job slot to run the job. You can use the **input and output sandboxes** to send small data files or scripts with your job.

More about Grid basics?

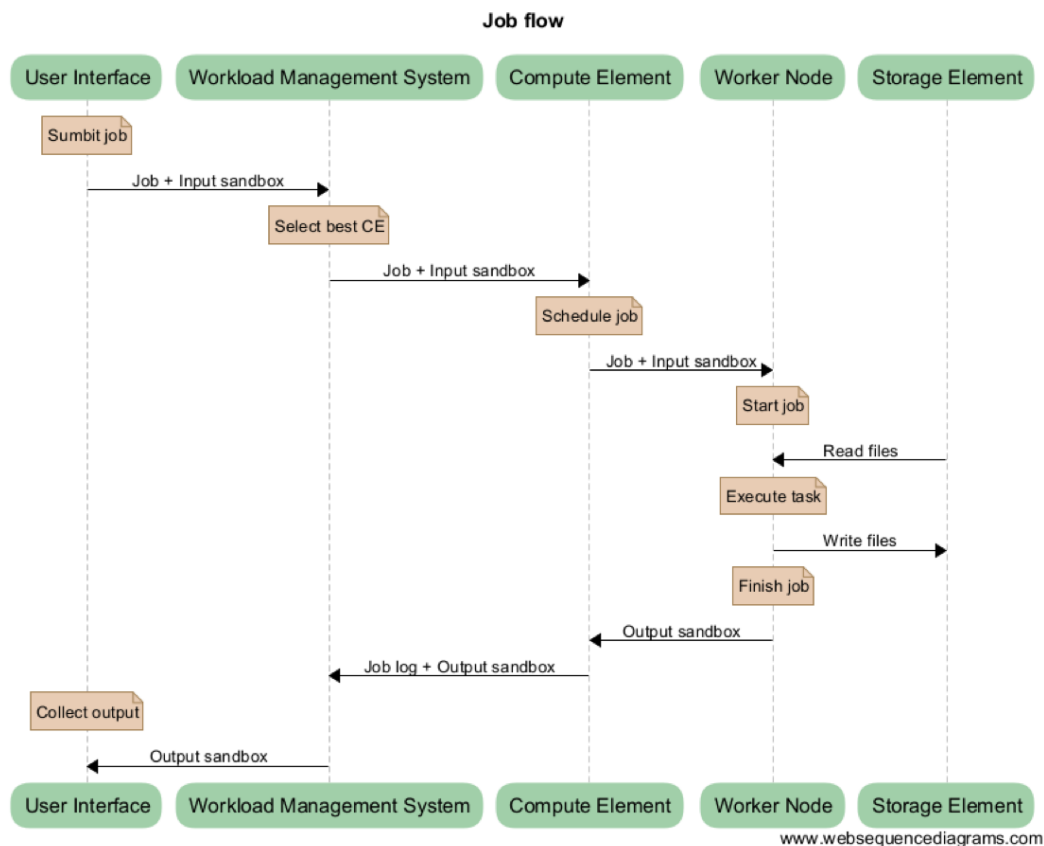
See also:

Check out our mooc video [Grid Computing Overview](#)

Each job is then submitted as a JDL (Job Description Language) file to the **Workload Management System** (WMS). The WMS is a *resource broker* that knows which Grid compute clusters are ready to accept your job and fulfil its requirements. Each Grid cluster consists of a **Compute Element** (CE) and several **Worker Nodes** (WNs). The CE (Compute Element) is a scheduler within each Grid cluster that communicates with the WMS (Workload Management System) about availability of the job slots in the cluster, and accepts and distributes the jobs to the available compute nodes in the cluster (these are called Worker Nodes or WNs). These WNs (Worker Nodes) are the machines which do the actual work. When finished with a job they will report back to the CE, which in turn will inform the WMS about the status of the job.

In addition, the Grid's interconnected clusters each have a storage server, called a **Storage Element** (SE), which can hold the input and output data of the jobs. Data on the SES (Storage Elements) can be replicated at multiple sites if needed for scale-out scenarios. In general, all SES offer disk storage for the staging of datasets before and after job execution. In addition, a central Grid storage facility (see [dCache](#)) also provides tape storage for long-term storage of datasets that need to be preserved.

In short, as a user you submit your jobs to execute your calculation or analysis code and to handle your input and output data. The WMS distributes the jobs to the clusters and node that are most suitable for these jobs. When the jobs are finished, you can collect the results from the SE (Storage Element) that was selected to hold the output data or keep them for later use on the central Grid storage facility.



1.1.3 To use the Grid or not

Grid computing is a form of distributed computing which can be *very powerful when applied correctly*. Grid is best suited for applications with a data-parallel nature that require many simultaneous *independent* jobs. With the help of the Grid, large scale computational problems can be solved and large amounts of data can be handled and stored.

Note: The Grid suits applications that can be split up relatively easily in multiple, independent parts or else **embarrassingly parallel** jobs.

Other HPC options

The Grid will be an interesting service if you are faced with workloads that concern hundreds of thousands of core hours and/or many terabytes of data. For Life Scientists we provide the [Life Science Grid](#) that offers additional functionality for smaller scale workloads. For other applications that concern small data or compute requirements, please have a look for other suitable [HPC systems](#) at SURFsara.

Job submission has a relatively high overhead. Submitting a “hello world” program may take minutes. Your data and your software must be available on the worker nodes, which requires careful planning of the job workflow. With the size of the job collections typical for the Grid, and submitting hundreds or even thousands jobs simultaneously, it may become a challenge to check your jobs for status and reschedule based on judgement of failures and their causes. We offer tools to help you automate these actions (see [Pilot jobs](#)), however, porting of your solution to the Grid will

always require time and effort to set up. Our experienced consultants are available for assistance and to help you make the right decisions right from the beginning.

The Grid infrastructure is able to accommodate a variety of communities and scientific fields, each with their own type of application and requirements, and without mutual interference. Typical Grid applications are:

- Massive data processing workloads.
- Large computational job collections that require a minimal time to completion.
- Projects that require collaboration and resource sharing with national or international partners.

1.2 Dutch National Grid

In this page you will find information about the Dutch National Grid Infrastructure:

Contents

- *Dutch National Grid*
 - *About*
 - *National and European*

1.2.1 About

The Grid in general consists of a large number and variety of clusters which are distributed all over the Netherlands and abroad. Grid infrastructures in European member states are organised by means of their own **National Grid Initiatives** (NGIs). SURFsara offers Grid services in collaboration with [Nikhef](#) and [RUG-CIT](#), and together we form the **Dutch National Grid Initiative** (NGI_NL).

The Dutch National Grid Initiative provides the Grid infrastructure to scientists affiliated with Dutch universities and other research institutes, and to international research projects that have established agreements with us.

Grid technology can be used for a variety of purposes and application types, however a subset of the worldwide Grid infrastructure can be dedicated for a specific purpose. For example our [Dutch Grid](#) infrastructure, which consists of a small number of tightly interconnected clusters, operated under the umbrella of NGI_NL, is very potent in enabling *high-throughput processing of large datasets* in a minimum amount of time.

1.2.2 National and European

The Dutch National Grid Infrastructure is connected to [EGI](#), the European Grid Initiative, which allows Grid users in the Netherlands and abroad easy access to one another's resources. EGI also supplies central monitoring and security and ensures smooth interoperability of all connected clusters.

1.3 Life Science Grid

On this page you will find general information about the Life Science Grid (LSG), the users of the LSG infrastructure, and the available LSG clusters:

Contents

- *Life Science Grid*
 - *About*
 - *For whom*
 - *LSG clusters*

1.3.1 About

The **Life Science Grid (LSG)** is a network of compute clusters intended specifically for researchers in the life sciences. The LSG infrastructure consists of a series of connected computer clusters which are placed within the working environment of Life Scientists while being fully managed remotely by experts at SURFsara.

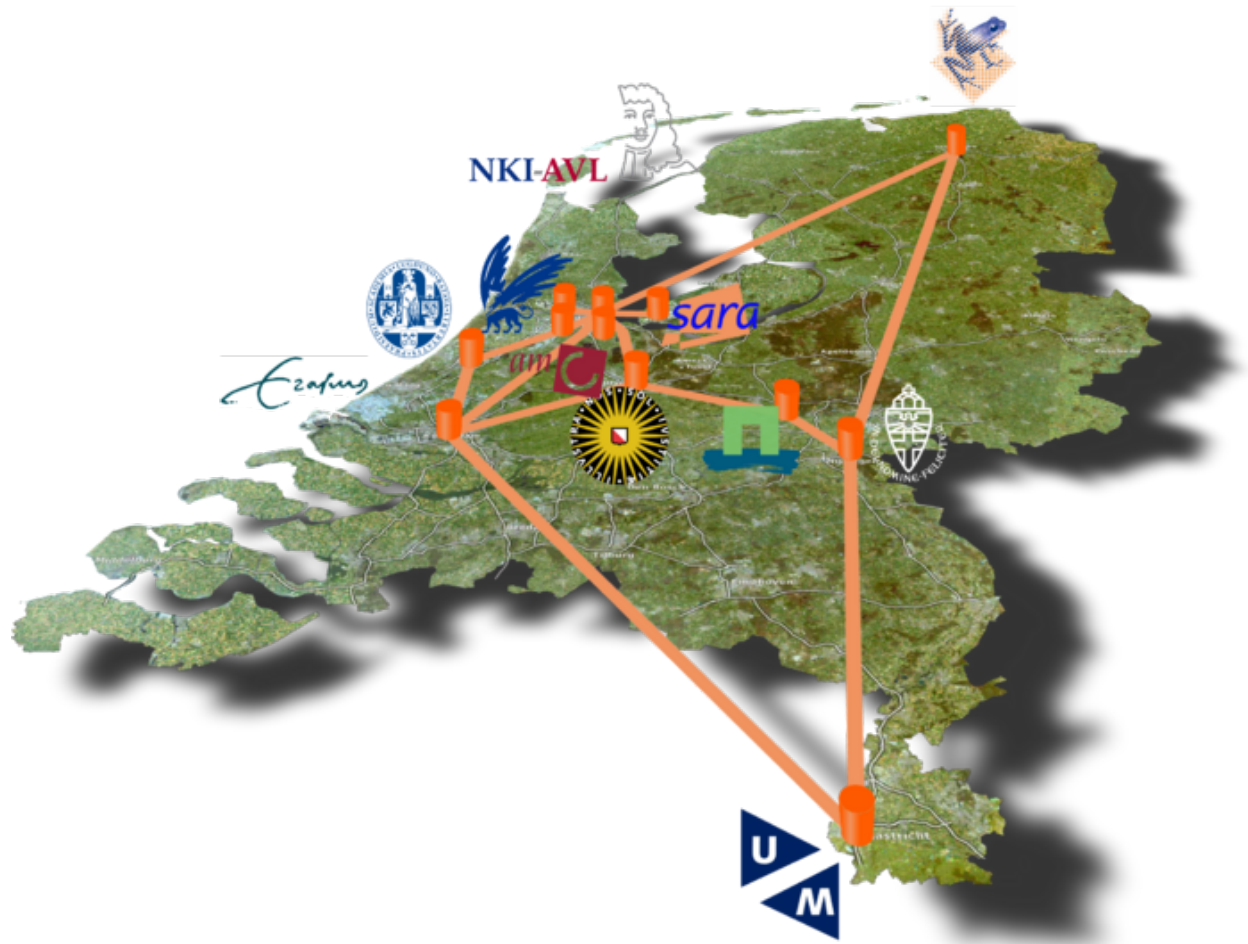
Since 2007, SURFsara has placed several powerful computer clusters at the local sites of interested universities. Research institutions in the Netherlands house clusters that are embedded in the international Grid infrastructure.

1.3.2 For whom

The Life Science Grid is open to all Life Scientists based in the Netherlands. It accommodates Life Scientists on Dutch universities and medical centers to perform data analysis or other computational work on a variety of scales, from a few occasional analysis runs up to thousands of production jobs continuously, on datasets ranging from a few gigabytes to hundreds of terabytes and beyond.

The LSG infrastructure is tailored specifically for applications within the life-sciences domain, with features that follow-up on specific difficulties experienced by today's Life Scientists. Data and applications can be shared not only among colleagues in your own research lab but also with collaborators at other locations, which is not a trivial thing in highly secured hospital environments. Data may also be secured behind the institute walls if needed.

1.3.3 LSG clusters



Currently, ten LSG (Life Science Grid) clusters are in place at the following sites:

LSG_BCBR	Utrecht
LSG_AMC	Amsterdam
LSG EMC	Rotterdam
LSG_KUN	Nijmegen
LSG_LUMC	Leiden
LSG_RUG	Groningen
LSG_TUD	Delft
LSG_UM	Maastricht
LSG_VU	Amsterdam
LSG_WUR	Wageningen

The technical specifications of a single LSG cluster are described in *LSG specifications*.

1.4 Grid services

In this page you will find general information about the SURFsara Grid services and support:

Contents

- *Grid services*
 - *About*
 - *Support*

1.4.1 About

To deploy your production tasks smoothly on the Grid, our services can be of great value. Our standard Grid services such as the Workload Management System, Compute Elements, Storage Elements, a Logical File Catalog, Virtual Organisation Management Services, and User Interface machines are indispensable for generic use of the Grid infrastructure. Apart from those we have additional services available as an option, that may be beneficial for your type of application.

Our Grid services provide:

- High-throughput processing nodes for job execution
- Access to scalable *Grid storage* facilities, disk and tape
- Standard Grid Services like User Interfaces, Brokers, and Virtual Organisation management services
- *Token Pool Services* (*ToPoS Overview* and *Picas Overview*) for production run logistics, job collection management, monitoring and control
- Virtual filesystem service *Softdrive* for centralized software deployment on distributed systems
- Dedicated *light paths*: we offer support bridging the last mile between the end points and your data sources.
- *Consultancy*: advice and support on getting access, working with Grid certificates, basic job submission, data access methods, best practices, advice on design and optimization of applications for performance improvements, integration with large-scale job-submission frameworks, and international upscaling.

1.4.2 Support

You may request support for our Grid services by contacting us by phone or email. Our dedicated team at [SURFsara helpdesk](#) is more than willing to assist you for any questions or complaints and carefully take note of your remarks for further improvement of our services.

Check out the detailed information about [SURFsara helpdesk](#). Please don't hesitate to contact us!

1.5 How to get access

In this page you will find general information about getting access to the Life Science Grid and the National Dutch Grid infrastructure:

Contents

- *How to get access*
 - *Local access to an LSG cluster*
 - *Access to the National Dutch Grid or LSG*
 - * *Estimate your resource needs*

1.5.1 Local access to an LSG cluster

Granting local access to a Life Science Grid (LSG) cluster enables you:

- submitting *PBS local jobs* directly to your local cluster
- the possibility to scale up to other LSG clusters (see *Access to the National Dutch Grid or LSG*)

If your research applies to one of the life science disciplines and your institute hosts a LSG cluster (see *LSG clusters*), you are eligible to obtain an account on your local LSG cluster.

To get an account on a local user interface, please send a request to your Designated Site Admin (see *Locations and local support contacts*) or contact us at helpdesk@surfsara.nl.

1.5.2 Access to the National Dutch Grid or LSG

Access to the Dutch Grid or the LSG clusters (including non-local access) allows for:

- submitting *Grid jobs* to multiple clusters via the Grid middleware
- storing data to the *Grid storage*

Researchers at SURF-affiliated institutes can apply for compute and storage capacity on the Grid by submitting the *SURFsara application form*. For scientists not affiliated to SURF, rates are based on tailor made packages. Specific details on obtaining accounts can be found in the *Access Grid* section of our website.

Please contact us at helpdesk@surfsara.nl for any inquiry on our possibilities.

Estimate your resource needs

When you request to use the Grid you do so in the context of a project. A project is set to solve a problem, defined as some goals that you want to achieve, which includes a plan on how you will achieve those goals. For each project a suitable amount of resources will be allocated. You can work together with several people in the same project and using the same resource allocation.

In general, each project resource allocation involves:

- an amount of compute time, measured in core-hours
- an amount of storage space, for disk or tape or both
- a start and end date for your project

In order for us to make a suitable allocation for a project, we need to know what your goals are and how you want to achieve those goals. That is why it is important for you to understand how to estimate the resources for your project.

not sure how to calculate your resource requirements?

Contact us at helpdesk@surfsara.nl and we can work together on estimating the resources for running your computation.

For a proper estimation of resources requirements it is best to start with a few test runs of the code (if existing) on another system (e.g.: your laptop). The outcome of such tests will in many cases be sufficient to derive a total requirement for your larger-scale production runs. Sometimes a more elaborate process is needed to come to an estimate, for example if the code does not exist yet or if changes to the existing code are still pending. Ideally you have been running a few representative samples of your runs before you file a resource request, in order to have some concrete information ready about the resources needed by your jobs, such as:

- how long it takes to run a representative input scenario

- how much space you need to store input, output and intermediate data
- what the software requirements are (required software tools, libraries, compilers, etc)
- how many scenarios (jobs) you need to run for a complete analysis

In case you don't know how to prepare such an inventory, we would be happy to assist you with that.

2.1 Prerequisites

This section summarises all the preparations you need to make before you run a *simple job* on the Grid:

Contents

- *Prerequisites*
 - *Preparation*
 - *Get a User Interface account*
 - *Get a Grid certificate*
 - *Join a Virtual Organisation*

2.1.1 Preparation

The Grid is a cooperation of many different clusters and research organisations, and as such, there is no centralised user management. Yet, there must be a way for the system to identify you and your work. This is why **Grid certificates** and **Virtual Organisations** (VOs) are introduced.

More about Grid prerequisites?

See also:

Check out our mooc video *Working Environment - Grid prerequisites*.

Your digital identity starts with a private key. **Only you** are allowed to know the contents of this key. Next, you need a Grid certificate, which is issued by a Certificate Authority (CA). The Grid certificate contains your name and your organisation, and it says that the person who owns the private key is really the person mentioned, and that this is certified by the Certificate Authority.

Now this is your identity. Big international cooperations do not want to deal with every user individually. Instead, users become part of Virtual Organisations. Individual clusters give access and compute time to certain VOs, and if you are a member of a VO (Virtual Organisation), you can run your jobs on that cluster.

More about Grid Security?

See also:

Check out our mooc video [Grid Certificate - Security](#).

In order to run your work on the Grid, you have to make three essential steps:

1. [Get a User Interface account](#), so that you can interact with the Grid.
2. [Get a Grid certificate](#), so that you can be identified on the Grid.
3. [Join a Virtual Organisation](#), so that you can run your jobs on the Grid.

The UI account will provide you with the proper environment to submit your jobs to the Grid. The Grid certificate is required to authorize you for using the Grid. Finally, the VO membership is based on your research domain (e.g. `lsgrid` for Life Scientists) and determines which resources you can use.

These steps are described in this chapter.

2.1.2 Get a User Interface account

In this section we will describe how to get a User Interface account.

The User Interface (UI) account will provide you with the environment to interact with the Grid. It is your access point to the Grid. You log in to a UI machine via SSH. On this computer you can, amongst others, do the following things: access Grid resources, create proxies, submit jobs, compile programs or prototype your application. For debugging purposes it is good to know that the environment of a user interface is similar to a Grid worker node thus if your code runs on the user interface it will most likely run on a Grid worker node.

To get this UI account, there are two options:

- If you work in Life Sciences sector and your local institutional cluster is part of the [Life Science Grid](#) (see [LSG clusters](#)), then you can get an account to the local user interface. Please send your request to your designated site admin (see [local contacts](#)) or contact us at helpdesk@surfsara.nl.
- At any other case, we will give you an account on the SURFsara Grid UI (server name: `ui.grid.sara.nl`). For this, you need to contact us at helpdesk@surfsara.nl.

Please note that the UI is simply a Linux machine and working on it requires some familiarity with linux commands and remote access methods. If you need help with this, check out our mooc video [Working Environment - Remote access](#).

2.1.3 Get a Grid certificate

Grid certificates are supplied by a Certificate Authority (CA). Users affiliated with Dutch institutes can request a digital certificate either by `DigiCert CA` or `DutchGrid CA`.

How to obtain a Grid certificate?

See also:

Find detailed info in our mooc video [Obtain a Grid Certificate](#).

If you are a researcher in the Netherlands we recommended you to request a certificate via `DigiCert CA`, by using your institutional login. This is the easiest and fastest way to get a Grid certificate. In cases that the `DigiCert CA` option is not applicable, you can request a `DutchGrid CA` certificate.

Here you can find details for obtaining and installing a Grid certificate:

DigiCert certificate

This section describes how to obtain and install a DigiCert Grid certificate. This is a prerequisite to get started on the Grid.

Contents

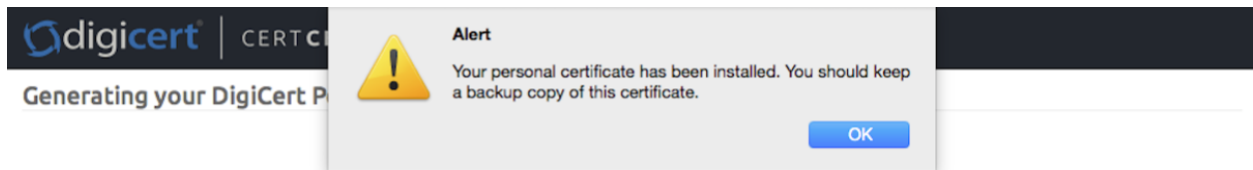
- *DigiCert certificate*
 - *Obtain a DigiCert certificate*
 - *Install a DigiCert certificate on the UI*
 - * *Export certificate from browser*
 - * *Copy certificate .p12 file to the UI*
 - * *Convert pkcs12 to PEM*
 - *Install a DigiCert certificate in your browser*

Note: If you need help to obtain your DigiCert certificate, please have a look to this [User Guide](#) or contact us at helpdesk@surfsara.nl.

Obtain a *DigiCert* certificate

DigiCert CA allows you to get your Grid certificate *instantly* from the GEANT Trusted Certificate Service (former was the Terena portal), by using your institutional login and SURFconext.

- Open a Firefox browser in your laptop or in your *UI account*
- Access the [DigiCert portal](#)
- Select your institution from the list and login with your account
- Request a so called Grid certificate. Select: **Product:** Grid Premium
- If everything went well, after a while you should see this window:



Once finished, you will have a Grid certificate automatically stored in your browser.

Install a *DigiCert* certificate on the UI

In order to install the *DigiCert* certificate on the UI, you need to export it first from your browser, copy it to your *UI account* and convert it to .pem format. This section shows you how to do this.

Export certificate from browser You can export the certificate from the browser that you stored your certificate in the previous step:

- Open the Firefox browser where the certificate is stored. This is the browser you used to access the [DigiCert portal](#)

- Select: Preferences -> Advanced (left pane) -> Certificates (tab) -> View Certificates (button)
- Select the certificate (.p12 file) that you stored in the previous step
- Press Backup
- Give it a name, e.g. browsercert and select the location to store it
- Give a safe password and press Ok

The file browsercert.p12 is now stored locally. Next, we will store it on the User Interface.

Copy certificate .p12 file to the UI

- Open a terminal and connect to the User Interface with your personal *UI account*:

```
$ssh homer@ui.grid.sara.nl # replace "homer" with your username!
```

- Create a \$HOME/.globus directory in your UI account:

```
$mkdir $HOME/.globus
```

- If you exported the certificate to your laptop, copy it from your local machine to your .globus directory on the UI. If you exported your certificate from the UI browser, you can skip this step:

```
[homer@localmachine]$scp /PATH-TO-P12-FILE/browsercert.p12 homer@ui.grid.sara.nl:~/.globus # re
```

Convert pkcs12 to PEM

- Convert the .p12 file to the PEM format. For this you need *two* commands; a) one to extract the key, and b) one to extract your certificate.

1. Extract your key, run on the UI:

```
$cd $HOME/.globus
$openssl pkcs12 -in browsercert.p12 -out userkey.pem -nocerts
```

Note that you will first need to enter the password that was used to *create* the browsercert.p12 file. Next, you need to enter a password to protect the exported key. Enter that password again to verify. Note that you must enter a password and the password must be at least 12 characters; if the password is too short, openssl will fail without error. Using the same password as for the p12 file is fine.

2. Extract your certificate, run on the UI:

```
$cd $HOME/.globus
$openssl pkcs12 -in browsercert.p12 -out usercert.pem -nokeys -clcerts
```

- Set the proper permissions to your certificate files:

```
$chmod 644 usercert.pem
$chmod 400 userkey.pem
```

The certificate and private key file should now be present in the .globus directory (notice the dot!) on the User Interface. Note that the private key file should be **read-only** and only readable to you.

- Verify key permissions:

```
$cd $HOME/.globus
$ls -l
```

-rw-r--r--	1	homer	homer	4499	May 10 13:47	usercert.pem
-r-----	1	homer	homer	963	May 10 13:43	userkey.pem

Install a *DigiCert* certificate in your browser

In order to apply for a *VO membership* you will have to install your certificate in your browser. If everything worked gracefully when you *obtained the DigiCert certificate* then your certificate was *automatically* stored in your browser.

- Verify that your certificate is valid and properly installed in your browser by accessing this website from the browser that you have your certificate installed:

<https://voms.grid.sara.nl:8443/vomses/>

If you receive an SSL authentication error, then try repeating the steps carefully as they come. If you managed to access the page above, your certificate is successfully installed!

See also:

Does my key match the certificate?

What is the expiry date of my certificate?

How can I see the subject of my certificate?

DutchGrid certificate

This section describes how to obtain and install a DutchGrid Grid certificate. This is a prerequisite to get started on the Grid:

Contents

- *DutchGrid certificate*
 - *Obtain a DutchGrid certificate*
 - * *Request a DutchGrid certificate*
 - * *Retrieve your DutchGrid certificate*
 - *Install a DutchGrid certificate on the UI*
 - *Install a DutchGrid certificate in your browser*
 - * *Convert PEM to pkcs12*
 - * *Import the certificate to the browser*

Obtain a DutchGrid certificate

In case that your institute does not support SURFconext and is not possible to get a *DigiCert certificate*, then you can apply for a DutchGrid CA certificate. You can request a DutchGrid certificate by launching the *JGridstart* tool.

Request a DutchGrid certificate

- Log in to your *UI account* with X forward enabled, e.g.:

```
$ssh -X homer@ui.grid.sara.nl # replace "homer" with your username!
```

- Download the the *jGridstart* tool:

```
$wget http://ca.dutchgrid.nl/start/jgridstart-wrapper-1.16.jar
```

- Run the wizard:

```
$java -jar jgridstart-wrapper-1.16.jar
```

- Follow the wizard instructions. You will typically go through these steps:
 - Start the Wizard by pressing `Request new . .` button
 - Generate request by entering your details (name, surname, email, organisation). At this stage you will provide the password for your Grid certificate - make sure you keep this safe!
 - Submit request. This will create your private `userkey.pem` file in your `~/ .globus` directory.
 - Fill in and print the verification form by pressing the `display form` button. Once you fill in the form, save it locally.
 - Close the wizard
- Check your details in the printed form and contact your institution's `Registration Authority (RA)` in person. The RA person will check your identity (id or passport or driving license) and sign the printed form.
- Once your form is signed by the RA (Registration Authority), send a scanned copy to the DutchGrid CA via email or fax. The contact details can be found in the printed form, but you can contact also helpdesk@surfsara.nl if you are in doubt.
- The DutchGrid CA (Certificate Authority) will finally send your certificate via email within ~a week. Once you have received your certificate you will need to install it both on your *UI account* and your browser (UI or laptop). We'll see this next.

Note: If you need help to obtain your DutchGrid certificate, please read the [JGridstart guide](#) or contact us at helpdesk@surfsara.nl.

Retrieve your DutchGrid certificate Once your request is approved, you will receive an email titled "*DutchGrid CA certificate ...*". Now you need to retrieve the new certificate:

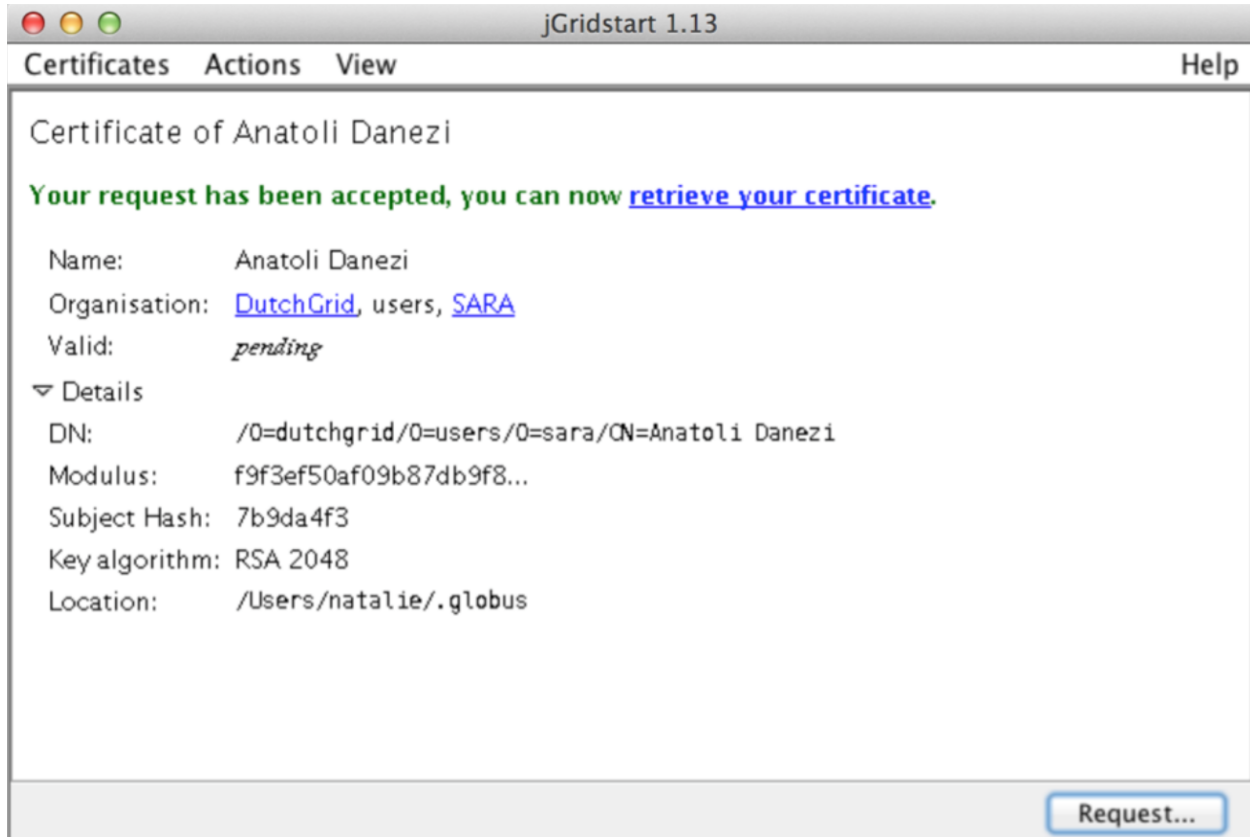
- Log in to your *UI account* with X forwarding enabled, e.g.:

```
$ssh -X homer@ui.grid.sara.nl # replace "homer" with your username!
```

- Run the wizard again:

```
$java -jar jgridstart-wrapper-1.16.jar
```

Then a window pops up similar to the following:



- Click on **retrieve your certificate**. This will automatically create a file `usercert.pem` in your `~/ .globus` directory (check with `$ ls ~/ .globus`).
- You may skip the step “install in browser” because the X session on the UI is slow and will probably be interrupted. Just click “Next”
- Close the wizard.

If everything went well, your certificate and key files (`usercert.pem` and `userkey.pem`) should be in the `~/ .globus` directory.

Install a DutchGrid certificate on the UI

If you followed the steps above properly, then your DutchGrid certificate and private key file should now be present in the `~/ .globus` directory (notice the dot!) on the User Interface machine. All you need to do is to set the proper permissions.

- Log in to your *UI account*:

```
$ssh homer@ui.grid.sara.nl # replace "homer" with your username!
```

- Set the proper permissions to your certificate files:

```
$cd $HOME/.globus
$chmod 644 usercert.pem
$chmod 400 userkey.pem
```

Note that the private key file should be **read-only** and only readable to you.

- Verify the correct permissions:

```
$ cd $HOME/.globus
$ ls -l
-rw-r--r--      1 homer    homer      4499  May 10 13:47  usercert.pem
-r-----      1 homer    homer        963  May 10 13:43  userkey.pem
```

Install a DutchGrid certificate in your browser

In order to apply for a *VO membership* you will have to install your certificate in your browser. Note that you can do this from any browser, however for convenience we will describe the procedure using the UI browser.

- Log in to your *UI account*:

```
$ssh homer@ui.grid.sara.nl # replace "homer" with your username!
$cd $HOME/.globus
```

Warning: You can import a certificate in your browser only when it is in the **PKCS12** format. This means that you need to convert the `usercert.pem` and `userkey.pem` files to a single `.p12` file.

Convert PEM to pkcs12

- To convert a PEM file to the PKCS12 format, run on the UI:

```
$openssl pkcs12 -export -inkey userkey.pem -in usercert.pem -out browsercert.p12
```

This will ask you for a password three times: the first is to unlock your private key stored in the file `userkey.pem`. The PKCS12-file will be password protected, which needs a new password, and the same password for confirmation. Note that you can use the same password as the password for the private key file, but this is not necessary.

Import the certificate to the browser

- To import the `.p12` file in your browser, open a Firefox window (`$ firefox &`) on the UI and apply the following steps (Note that you may have to copy the `.p12` file to a directory accessible from your browser):
 - From the Firefox Menu bar select: Edit > Preferences > Encryption > View Certificates > Import
 - Select the `browsercert.p12` file from the UI local directory
 - Give the password you set in the previous step.
 - You should now see the certificate listed. Close the window.

Problems installing the certificate?

See also:

Need more details for installing your certificate on the UI or browser? Check out our mooc video *User Interface machine*.

- Verify that your certificate is valid and properly installed in your browser by accessing this website:

<https://voms.grid.sara.nl:8443/vomses/>

If you receive an SSL authentication error, then try repeating the steps carefully as they come. If you managed to access the page above, your certificate is successfully installed!

See also:*Does my key match the certificate?**What is the expiry date of my certificate?**How can I see the subject of my certificate?*

2.1.4 Join a Virtual Organisation

More about VOs?**See also:**Need to know more about VOs and how to get a membership? Check out our mooc video *Virtual Organisations*.

A Virtual Organisation or VO is a group of geographically distributed people that have common objectives and that are using shared Grid resources to achieve them. Every Grid user is a member of one or more Virtual Organisations.

In practice your VO membership determines to which resources (compute and storage) you have access to. You are eligible to register for a VO only once you *get a valid certificate*. The VO that is most suitable for you depends on the specific research area you are in. For example, if you are active in a field associated with the life sciences the `lsgrid` VO might be most suitable for you. If you still not sure which VO is best for you, then contact us at helpdesk@surfsara.nl to guide you on this.

This section describes how to get a VO membership.

Warning: At this point you must have the certificate successfully installed in your browser. If you don't have that, go back the *previous step*.

- Open the link below from the browser where your certificate is installed (UI or laptop). The page will ask you to confirm with your installed browser certificate: <https://voms.grid.sara.nl:8443/vomses/>
- Select the VO that you are interested for e.g. `lsgrid`, from the front page listing all the available VOs (Virtual Organisations). If it is unclear to you for which VO you should register, please contact us at helpdesk@surfsara.nl.
- Fill in some of your personal details (name, email, etc.), then read and accept the AUP (Acceptable Use Policy).
- Check that you have received a verification email titled “Your membership request for VO ...” and confirm with the URL, as described in the email.
- You will receive an approval email titled “Your vo membership request for VO `lsgrid` has been approved” when the VO administrator finally signs your request.

Once you finish this page instructions successfully, you are set to go and run your *First Grid job*!

2.2 First Grid job

This section summarises all the steps to submit your first job on the Grid, check its status and retrieve the output:

Contents

- *First Grid job*
 - *Grid job lifecycle*
 - *StartGridSession*
 - *Describe your job in a JDL file*
 - * *Job list match*
 - *Submit the job to the Grid*
 - *Track the job status*
 - * *Cancel job*
 - *Retrieve the output*
 - * *Check job output*
 - *Recap & Next Steps*

Warning: You can continue with this guide *only after* you have completed the *preparations* for Grid. If you skipped that, go back to the *Prerequisites* section. Still need help with obtaining or installing your certificate? We can help! Contact us at helpdesk@surfsara.nl.

Once you finish with the *First Grid job*, you can continue with more *advanced topics* and also *Best practices*, the section that contains guidelines for porting real complex simulations on the Grid.

2.2.1 Grid job lifecycle

Grid job lifecycle

See also:

Have a look at our mooc video that describes the *Grid job Lifecycle* step by step.

To run your application on the Grid you need to describe its requirements in a specific language called **job description language (JDL)**. This is similar to the information that we need to specify when we run jobs using a batch scheduling system like *PBS local jobs*, although it is slightly more complex as we are now scheduling jobs across multiple sites.

Except for the application requirements, you also need to specify in the JDL the content of the *input/output sandboxes*. These sandboxes allow you to transfer data to or from the Grid. The input sandbox contains all the files that you want to send with your job to the worker node, like e.g. a script that you want executed. The output sandbox contains all the files that you want to have transferred back to the UI.

Note: The amount of data that you can transfer using the sandboxes is very limited, in the order of a few megabytes (less than **100MB**). This means that you should normally limit the input sandbox to a few script files and the output sandbox to the stderr and stdout files.

Once you have the JDL file ready, you can submit it to multiple clusters with `glite-*` commands. The Workload Management System (WMS) will schedule your job on a Grid worker node. The purpose of WMS is to distribute and manage tasks across computing resources. More specifically, the WMS will accept your job, assign it to the most appropriate Computing Element (CE), record the job status and retrieve the output.

The following animations illustrate the Grid lifecycle as described above:

- [Grid WMS animation](#)
- [Grid job status animation](#)

2.2.2 StartGridSession

Before submitting your first Grid job, you need to create a *proxy* from your certificate. This has a short lifetime and prevents you from passing along your personal certificate to the Grid. The job will keep a copy of your proxy and pass it along to the Worker Node.

This section will show you how to create a valid proxy:

- Log in to your UI account:

```
$ssh homer@ui.grid.sara.nl # replace "homer" with your username
```

- Create a proxy with the following command and provide your Grid certificate password when prompted:

```
$startGridSession lsgrid #replace lsgrid with your VO
```

You should see a similar output displayed in your terminal:

```
Now starting...
Please enter your GRID password:
voms-proxy-init -voms lsgrid --valid 168:00 -pwstdin
Contacting voms.grid.sara.nl:30018 [/O=dutchgrid/O=hosts/OU=sara.nl/CN=voms.grid.sara.nl] "lsgrid"
Remote VOMS server contacted successfully.

Created proxy in /tmp/x509up_u39111.

Your proxy is valid until Tue Jan 11 09:31:56 CET 2016
Your identity: /O=dutchgrid/O=users/O=sara/CN=Homer Simpson
Creating proxy ..... Done
Proxy Verify OK
Your proxy is valid until: Tue Jan 11 09:31:56 2016
A proxy valid for 168 hours (7.0 days) for user /O=dutchgrid/O=users/O=sara/CN=Homer Simpson now
Your delegation ID is: homer
```

Note: What does the startGridSession script actually do?

- It generates a *local proxy* x509up_uXXX in the UI /tmp/ directory
- It uploads this proxy to Myproxy server
- It delegates the proxy to the WMS with your user name as the delegation ID (DID)

If you want to know more, see the advanced section about [Grid authentication](#).

And now you are ready to submit jobs to the Grid! Or copy data from and to the Grid.

2.2.3 Describe your job in a JDL file

To submit a Grid job you must describe this in a plain text file, called JDL. Optionally, you can check the Computing Elements (CEs) that this job may run on. The JDL file will pass the details of your job to the WMS.

Warning: Make sure you have started your session and created already a *valid proxy*.

- Log in to your User Interface.
- Create a file with the following content describing the job requirements. Save it as `simple.jdl`:

```
1  Type = "Job";
2  JobType = "Normal";
3  Executable = "/bin/hostname";
4  Arguments = "-f";
5  StdOutput = "simple.out";
6  StdError = "simple.err";
7  OutputSandbox = {"simple.out", "simple.err"};
```

This job involves no large input or output files. It will return to the user the hostname of the Worker Node that the job will land on. This is specified as the StdOutput file `simple.out` declared in the OutputSandbox statement.

Job list match

Before actually submitting the job, you can optionally check the matching Computing Elements that satisfy your job description. It does not guarantee anything about the CE load, just matches your JDL criteria with the available VO resources:

```
$glite-wms-job-list-match -a simple.jdl # replace simple.jdl with your JDL file
```

Alternatively, use your delegation ID:

```
$glite-wms-job-list-match -d homer simple.jdl # replace homer with your delegation id, in this case y
```

Note: The `-a` option should not be used frequently. It creates a proxy of your certificate ‘on-the-fly’ when the job is submitted; therefore `-a` is quite inefficient when submitting hundreds of jobs.

Your job is now ready. Continue to the next step to submit it to the Grid!

To submit your first Grid job and get an understanding of the job lifecycle, we will perform these steps:

- *Job submission*
- *Status tracking*
- *Output retrieval*

2.2.4 Submit the job to the Grid

First Job explained

See also:

For more detailed information about submitting a simple Grid job, have a look at our mooc video [My First Grid job](#).

You should have your `simple.jdl` file ready in your UI up to this point. When you submit this simple Grid job to the WMS, a job will be created and sent to a remote Worker Node. There it will execute the command `/bin/hostname -f` and write its standard output and its standard error in the `simple.out` and `simple.err` respectively.

- Submit the simple job by typing in your UI terminal this command:

```
$glite-wms-job-submit -d $USER -o jobIds simple.jdl

Connecting to the service https://wms2.grid.sara.nl:7443/glite_wms_wmproxy_server
===== glite-wms-job-submit Success =====
```

```
The job has been successfully submitted to the WMPProxy
Your job identifier is:
```

```
https://wms2.grid.sara.nl:9000/JIVYfkMxtnRFWweGsx0XAA
```

```
The job identifier has been saved in the following file:
/home/homer/jobIds
=====
```

Note the use of `-d $USER` to tell your job that it should use your delegated proxy certificate.

The option `-o` allows you to specify a file (in this case `jobIds`) to store the unique job identifier:

- You can use this URL identifier to monitor your job from the command line or your browser and to get the job output.
- Note that omitting the `-o` option means that the jobID is not saved in a file. When you do not save this id you will effectively lose the output of your job!

The jobID string looks like this:

```
$cat jobIds

###Submitted Job Ids###
https://wms2.grid.sara.nl:9000/JIVYfkMxtnRFWweGsx0XAA
```

2.2.5 Track the job status

To check the current job status from the command line, apply the following command that queries the WMS for the status of the job.

- After submitting the job, type:

```
$glite-wms-job-status https://wms2.grid.sara.nl:9000/JIVYfkMxtnRFWweGsx0XAA #replace with your j
```

- Alternatively, if you have saved your jobIds into a file you can use the `-i` option and the filename as argument:

```
$glite-wms-job-status -i jobIds
```

- Finally, a third (optional) way to check the job status is with the web browser in which *you installed your certificate*. In this browser open the jobID link:

```
https://wms2.grid.sara.nl:9000/JIVYfkMxtnRFWweGsx0XAA #replace with your jobID
```

Note that the URL can only be accessed by you as you are authenticated to the server with the certificate installed in this browser. If your certificate is not installed in this browser, you will get an authentication error.

Cancel job

- If you realize that you need to cancel a submitted job, use the following command:

```
$glite-wms-job-cancel https://wms2.grid.sara.nl:9000/JIVYfkMxtnRFWweGsx0XAA #replace with your j
```

- Alternatively, you can use the `jobIds` file:

```
$glite-wms-job-cancel -i jobIds
```

2.2.6 Retrieve the output

The output consists of the files included in the `OutputSandbox` statement. You can retrieve the job output once it is successfully completed, in other words the job status has changed from `RUNNING` to `DONE`. The files in the output sandbox can be downloaded for approximately one week after the job finishes.

Note: You can choose the output directory with the `--dir` option. If you do not use this option then the output will be copied under the `UI /scratch` directory with a name based on the ID of the job.

- To get the output, type:

```
$glite-wms-job-output https://wms2.grid.sara.nl:9000/JIVYfkMxtnRFWweGsx0XAA #replace with your j
```

- Alternatively, you can use the jobIDs file:

```
$glite-wms-job-output --dir . -i jobIds
```

where you should substitute `jobIds` with the file that you used to store the job ids.

If you omitted the `--dir` option, your output stored on the `/scratch` directory on the UI. Please remove your files from the `/scratch` directory when they are no longer necessary. Also keep in mind that if the `/scratch` directory becomes too full, the administrators remove the older files until enough space is available again.

Check job output

- To check your job output, browse into the downloaded output directory. This includes the `simple.out`, `simple.err` files specified in the `OutputSandbox` statement:

```
$ls -l /home/homer/homer_JIVYfkMxtnRFWweGsx0XAA/
-rw-rw-r-- 1 homer homer 0 Jan 5 18:06 simple.err
-rw-rw-r-- 1 homer homer 20 Jan 5 18:06 simple.out

$cat /home/homer/homer_JIVYfkMxtnRFWweGsx0XAA/simple.out # displays the hostname of the Grid wor
wn01.lsg.bcbr.uu.nl
```

2.2.7 Recap & Next Steps

Congratulations! You have just executed your first job to the Grid!

Let's summarise what we've seen so far.

You interact with the Grid via the UI machine `ui.grid.sara.nl`. You describe each job in a JDL (Job Description Language) file where you list which program should be executed and what are the worker node requirements. From the UI, you create first a proxy of your Grid certificate and submit your job with `glite-*` commands. The resource broker, called WMS (short for Workload Management System), accepts your jobs, assigns them to the most appropriate CE (Computing Element), records the jobs statuses and retrieves the output.

This is a short overview of the commands needed to handle simple jobs:

startGridSession	startGridSession lsgrid
submit job	glite-wms-job-submit -d \$USER -o jobIds simple.jdl
job status	glite-wms-job-status -i jobIds
cancel job	glite-wms-job-cancel -i jobIds
retrieve job output	glite-wms-job-output --dir -i jobIds

See also:

Try now to port your own application to the Grid. Check out the *Best practices* section and run the example that suits your use case. The section *Advanced topics* will help your understanding for several Grid modules used in the *Best practices*.

Done with the *General*, but not sure how to proceed? We can help! Contact us at helpdesk@surfsara.nl.

Advanced topics

3.1 Grid software

In this page we will talk about the options to run your software on the Grid worker nodes:

Contents

- *Grid software*
 - *Softdrive*
 - * *CVMFS*
 - * *Quickstart*
 - *CVMFS group membership*
 - *Logging in on the softdrive*
 - *Distributing an example file*
 - *Finding your files on the Grid nodes*
 - *Python on the Grid*
 - * *Softdrive anaconda*
 - *Docker*

3.1.1 Softdrive

Softdrive is the service that allows you to install software in a central place and distribute it *automagically* on the Grid. You install the software once, and it will be available on all clusters, to all users. This means that you no longer need to supply your software in your *input sandbox*, or download your software in your job.

CVMFS

Softdrive is using the CVMFS (short for CernVM File System) tool on the background. CVMFS is a network file system based on HTTP and optimized to deliver experiment software in a fast, scalable, and reliable way.

Quickstart

In this example, we will distribute a few small files to all nodes in the Life Sciences Grid. This should give you an idea of what is possible with *Softdrive*.

Softdrive works by logging in the software distribution node, and putting your files there. Next, you tell the software distribution system that you are ready installing files. These files will be made available on all nodes in the *Life Science Grid* and on all other nodes on the *Dutch National Grid*.

CVMFS group membership

To distribute your files using Softdrive, you must be a member of the group `softdrive`. If you are not a member yet, send an e-mail to helpdesk@surfsara.nl with a membership request.

Logging in on the softdrive

After you have been added to the `softdrive` group, you can log in on the software distribution node, using your Grid UI username and password:

```
$ssh homer@softdrive.grid.sara.nl # replace homer with your username
```

In your home-directory (e.g. `/home/homer`), you will find a *README* file with detailed information about the *Softdrive* usage.

Distributing an example file

To demonstrate distributing files to all Grid nodes, create a file and a directory within your home directory:

```
# a test directory and a file
softdrive.grid.sara.nl:/home/homer$ mkdir -p test_dir
softdrive.grid.sara.nl:/home/homer$ echo "Hello world" > test_dir/hello.txt
```

To make this directory file available on all nodes on the Grid, you have to copy the `test_dir` under `/cvmfs/softdrive.nl/$USER`:

```
softdrive.grid.sara.nl:/home/homer$ cp -r test_dir /cvmfs/softdrive.nl/homer # replace homer with your username
```

- To force the update everywhere in the Grid, trigger publication by executing command:

```
publish-my-softdrive
```

Updating on all Grid nodes can take up to two hours.

Note: You need to run the command `publish-my-softdrive` each time you make a change in your `/cvmfs/softdrive.nl/$USER` directory in order to take effect on the Grid sites.

Finding your files on the Grid nodes

On nodes, your Softdrive files will be available under:

```
/cvmfs/softdrive.nl/homer/ # replace homer with your username
```

Log in to your *UI account* and check whether your files are there:

```
ui.grid.sara.nl:/home/homer$ ls /cvmfs/softdrive.nl/homer/
drwxr-xr-x 17 cvmfs cvmfs 4096 Dec 16 12:11 test_dir
```

Note: If your software is statically compiled, then copying the executables from your home directory to `/cvmfs/softdrive.nl/$USER/` should work. Just remember to export the `/cvmfs/softdrive.nl/$USER` software paths into your Grid scripts or UI `.bashrc`. In other cases with library path dependencies, we advice you to install your software directly under `/cvmfs/softdrive.nl/$USER` or use a prefix. An example of software installation in Softdrive can be found in section [anaconda on Grid](#).

3.1.2 Python on the Grid

On the local Grid clusters the python version installed is *Python 2.6.6*. If you need a different python version or additional packages, we recommend you to install [Anaconda python](#) in your UI or *Softdrive* account.

Next is an example of installing the *Anaconda* python distribution in *Softdrive*.

Softdrive anaconda

- Log in to Softdrive with your account:

```
$ssh homer@softdrive.grid.sara.nl # replace homer with your username
```

- Download in your home account the latest version of Anaconda installer for linux, e.g.:

```
$wget https://3230d63b5fc54e62148e-c95ac804525aac4b6dba79b00b39d1d3.ssl.cf1.rackcdn.com/Anaconda2-2.4.0-Linux-x86_64.sh
```

- Run the installer (read and approve the license terms) in Softdrive:

```
$bash Anaconda2-2.4.0-Linux-x86_64.sh
```

Note here! The installer will ask you to which location to install the software. Do not accept the default but change it to: `/cvmfs/softdrive.nl/$USER/anaconda-2-2.4.0/`:

```
Anaconda2 will now be installed into this location:
/home/homer/anaconda2
- Press ENTER to confirm the location
- Press CTRL-C to abort the installation
- Or specify a different location below

[/home/homer/anaconda2] >>> /cvmfs/softdrive.nl/homer/anaconda-2-2.4.0/
...
```

That was it! You can now publish the software that is installed in your `/cvmfs/softdrive.nl/homer/anaconda-2-2.4.0` directory. To do so, run this command in Softdrive:

```
$publish-my-softdrive
```

Then check after 1-2 hours from the UI if the `/cvmfs/softdrive.nl/homer/anaconda-2-2.4.0` exists.

Finally, remember to include the installation path in your scripts as:

```
$export PATH=/cvmfs/softdrive.nl/homer/anaconda-2-2.4.0/bin:$PATH # replace homer with your username
```

3.1.3 Docker

At the moment it is not possible to run Docker containers on the *Dutch National Grid* or *Life Science Grid*. We are currently investigating different possibilities. Please contact us at helpdesk@surfsara.nl to discuss about the available

options.

3.2 Grid storage

In this page we will talk about the Grid storage facilities, the tools to interact with it and the method to handle data that is stored on tape.

Contents

- *Grid storage*
 - *About Grid storage*
 - *Storage types*
 - * *dCache*
 - * *DPM*
 - *Grid file identifiers*
 - * *Transport URL or TURL*
 - * *Storage URL or SURL*
 - * *Logical File Name (LFN) and Grid Unique Identifier (GUID)*
 - * *Default ports*
 - *dCache*
 - *DPM*
 - *Storage clients*
 - *Staging files*
 - * *Staging a single file*
 - * *Staging groups of files*
 - * *Monitor staging activity*
 - * *Unpin a file*
 - *SRM interaction example diagram*
 - *Importing large amounts of data*

3.2.1 About Grid storage

Each cluster on the Grid is equipped with a Storage Element or SE where data is stored. The Grid storage is useful for applications that handle large amount of data that can not be sent with the *job Sandbox* or stored in a *pilot job database*.

You can interact with the Grid storage from the UI or from a Worker Node, within your running job. The scripts that can access the Grid storage can be submitted from:

- *The UI*
- *Any local LSG cluster*
- *The Dutch Grid*

To use the Grid storage you must:

- Have *a personal Grid certificate* [\[1\]](#)
- Be member of *a VO* for which we have allocated storage space.

You can access the Grid storage with Grid *Storage clients*, through interfaces that speak protocols like SRM (Storage Resource Management), GRIDFTP (File Transfer Protocol with Grid authentication), GSIDCAP (dCache Access

Protocol with Grid authentication) or WEBDAV (Web Distributed Authoring and Versioning). With these storage clients you can:

- list directories and files
- read (download) files
- write (upload) files
- delete files or directories
- stage files (copy them from tape to disk for faster reading)

3.2.2 Storage types

There are two storage types available on the Dutch Grid sites:

- The *dCache* storage element located at SURFsara and accessible from *any* Grid site.
- The *DPM* storage elements located at each *LSG cluster* and accessible *only* by the *Life Science Grid* users.

dCache

The storage element located at SURFsara is accessible from *any* Grid cluster or UI. It uses the *dCache system* for storing and retrieving huge amounts of data, distributed among a large number of server nodes. It consists of magnetic tape storage and hard disk storage and both are addressed by a common file system. See *dCache specifications* for details about our dCache instance.

DPM

The storage elements located at the various *Life Science Grid clusters* are accessible *only* by the LSG users. The LSG clusters have local storage that uses DPM (short for Disk Pool Manager).

Note: The DPM (Disk Pool Manager) storage is only disk storage and does not support tape back-end. In opposite, the dCache central storage has both disk and tape.

3.2.3 Grid file identifiers

You can refer to your files on the Grid with different ways depending on which of the available *Storage clients* you use to manage your files:

Transport URL or TURL

Examples:

```
# lsgrid user homer stores the file zap.tar on dCache storage
gsiftp://gridftp.grid.sara.nl:2811/pnfs/grid.sara.nl/data/lsgrid/homer/zap.tar

# same, but with a WebDAV TURL
https://webdav.grid.sara.nl/pnfs/grid.sara.nl/data/lsgrid/homer/zap.tar

# lsgrid user homer stores the file zap.tar on DPM storage at lumc cluster
gsiftp://gb-se-lumc.lumc.nl:2811/dpm/lumc.nl/home/lsgrid/homer/zap.tar
```

Clients for TURLs

- uberftp
- globus
- gfal
- fts
- globusonline

Storage URL or SURL

Examples:

```
# lsguid user homer stores the file zap.tar on dCache storage
srm://srm.grid.sara.nl:8443/pnfs/grid.sara.nl/data/lsguid/homer/zap.tar

# lsguid user homer stores the file zap.tar on DPM storage at lumc cluster
srm://gb-se-lumc.lumc.nl:8446/dpm/lumc.nl/home/lsguid/homer/zap.tar
```

Clients for SURLs

- srm
- gfal
- fts
- lcg-lfn-lfc

Logical File Name (LFN) and Grid Unique Identifier (GUID)

These identifiers correspond to logical filename such as `lfn:/grid/lsguid/homer/zap.tar`

Note: The SURLs (Storage URLs) and TURLs (Transport URLs) contain information about where a **physical** file is located. In contrast, the GUIDs (Grid Unique Identifiers) and LFNs (Logical File Name) identify a **logical** filename irrespective of its location. You only need to use these if you work with *Data replication* on multiple LSG sites.

Default ports**dCache**

Protocol	Host(s) and port(s)	Remark
SRM	srm://srm.grid.sara.nl:8443	
GridFTP	gsiftp://gridftp.grid.sara.nl:2811	Data channel port range: 20000-25000
WebDAV	https://webdav.grid.sara.nl:443	See <i>webdav client</i> for details
	https://webdav.grid.sara.nl:2880	
	https://webdav.grid.sara.nl:2881	
GSIdCap	gsidcap://gsidcap.grid.sara.nl:22128	
xroot	xrootd.grid.sara.nl:1094	Used by CERN only

DPM

Protocol	Host(s) and port(s) (examples)	Remark
SRM	srn://gb-se-lumc.lumc.nl:8446	
GridFTP	gsiftp://gb-se-lumc.lumc.nl:2811	Data channel port range: 20000-25000

For an overview of all life science clusters and their DPM storage elements, see [Cluster details](#)

3.2.4 Storage clients

The `InputSandbox` and `OutputSandbox` attributes in the *JDL* file are the basic way to move files to and from the User Interface (UI) and the Worker Node (WN). However, when you have large files (from about 100 MB and larger) then you should not use these sandboxes to move data around. Instead you should use the *Storage types* and work with several *Storage clients*.

In this section we will show the common commands to use the various storage clients.

Note: From the many Grid storage clients, we recommend you to use the *uberftp client*, *globus client* or *gfal client*. These tools have a clean interface, and their speed is much better on our systems compared with their *srn*-* equivalents.

Table 3.1: Storage clients

Client	protocols				3rd party	Speed	Tape control [1]
	SRM	GridFTP	GSId-Cap	Web-DAV			
<i>uberftp client</i>	–	yes	–	–	–	high	–
<i>globus client</i>	–	yes	–	–	–	high	–
<i>srn client</i>	yes	²	²	²	–		yes
<i>gfal client</i>	yes	yes	–	–	–		yes
<i>webdav client</i>	–	–	–	yes	–		–
<i>fts client</i>	yes	yes	–	yes	yes	high	yes
<i>globusonline client</i>	–	yes	–	–	yes	high	–
<i>lcn-lfn-lfc clients</i> (not recommended)	yes	²	–	–	–		–

uberftp client

This page includes the basic commands to use *uberftp*. For an overview of storage clients, see [Storage clients](#).

Contents

- *uberftp client*
 - *Uberftp*
 - * *Creating/listing*
 - * *Transferring data*
 - *Parallel streams*
 - * *Removing data*

²SRM and LCG commands use the SRM protocol for metadata level operations and switch to another protocol like GridFTP for file transfers. This may cause protocol overhead. For example, authentication needs to be done twice: once for each protocol.

Ubertftp

Creating/listing

Note: To run the examples below you need to have a valid proxy, see [StartGridSession](#).

- Listing directories on dCache:

```
$uberftp -ls gsiftp://gridftp.grid.sara.nl:2811/pnfs/grid.sara.nl/data/lsgrid/
```

- Listing directories on DPM:

```
$uberftp -ls gsiftp://gb-se-lumc.lumc.nl:2811/dpm/lumc.nl/home/lsgrid
```

- Create a new directory on dCache:

```
$uberftp -mkdir gsiftp://gridftp.grid.sara.nl:2811/pnfs/grid.sara.nl/data/lsgrid/homer/newdir
```

- Create a new directory on DPM:

```
$uberftp -mkdir gsiftp://gb-se-lumc.lumc.nl:2811/dpm/lumc.nl/home/lsgrid/homer/newdir
```

Transferring data

- Copy file from dCache to local machine:

```
$uberftp gsiftp://gridftp.grid.sara.nl:2811/pnfs/grid.sara.nl/data/lsgrid/homer/zap.tar file:///
```

- Copy file from DPM to local machine:

```
$uberftp gsiftp://gb-se-lumc.lumc.nl:2811/dpm/lumc.nl/home/lsgrid/homer/zap.tar file:///home/h
```

- Copy file from local machine to dCache:

```
$uberftp file:///home/homer/zap.tar gsiftp://gridftp.grid.sara.nl:2811/pnfs/grid.sara.nl/data/ls
```

- Copy file from local machine to DPM:

```
$uberftp file:///home/homer/zap.tar gsiftp://gb-se-lumc.lumc.nl:2811/dpm/lumc.nl/home/lsgrid/hom
```

Note: The asterisk “*” wildcard (match all characters) works with ubertftp. Please use this option with caution, especially when deleting files.

Parallel streams The GridFTP protocol allows for parallel streaming of data transfers. This makes a transfer more efficient and less susceptible to network errors, especially over long distances. If you have a lot of simultaneous transfers running anyway, increasing the number of streams per transfer will not make a big difference, because the network bandwidth may limit the results.

```
$uberftp -parallel 4 \  
$ gsiftp://gb-se-lumc.lumc.nl:2811/dpm/lumc.nl/home/lsgrid/homer/zap.tar \  
$ file:zap.tar
```

Results may vary based on circumstances. We suggest a number of 4 streams as a start.

Removing data

- Remove a file from dCache:

```
$uberftp -rm gsiftp://gridftp.grid.sara.nl:2811/pnfs/grid.sara.nl/data/lsgrid/homer/zap.tar
```

- Remove a file from DPM:

```
$uberftp -rm gsiftp://gb-se-lumc.lumc.nl:2811/dpm/lumc.nl/home/lsgrid/homer/zap.tar
```

- Remove whole (non-empty) directory with all content from dCache:

```
$uberftp -rm -r gsiftp://gridftp.grid.sara.nl:2811/pnfs/grid.sara.nl/data/lsgrid/homer/testdir/
```

- Remove whole (non-empty) directory with all content from DPM:

```
$uberftp -rm -r gsiftp://gb-se-lumc.lumc.nl:2811/dpm/lumc.nl/home/lsgrid/homer/testdir/
```

globus client

This page includes the basic commands to use `globus`. For an overview of storage clients, see [Storage clients](#).

Contents

- [globus client](#)
 - *Globus tools*
 - * *Creating/listing*
 - * *Transferring data*
 - *Parallel streams*
 - * *Removing data*

Globus tools

Note: To run the examples below you need to have a valid proxy, see [StartGridSession](#).

Creating/listing The `globus-*` client does not offer an option to create or list directories. For this purpose, use a different client, e.g. [uberftp client](#).

Transferring data

Note: The options `-dbg -gt 2 -vb` would show you extra logging information for your transfer.

- Copy file from dCache to local machine:

```
$globus-url-copy \
$ gsiftp://gridftp.grid.sara.nl:2811/pnfs/grid.sara.nl/data/lsgrid/homer/zap.tar \
$ file:///`pwd`/zap.tar
```

- Copy file from DPM to local machine:

```
$globus-url-copy \  
$ gsiftp://gb-se-lumc.lumc.nl:2811/dpm/lumc.nl/home/lsgrid/homer/zap.tar \  
$ file:///`pwd`/zap.tar
```

- Copy file from local machine to dCache:

```
$globus-url-copy \  
$ file:///`pwd`/zap.tar \  
$ gsiftp://gridftp.grid.sara.nl:2811/pnfs/grid.sara.nl/data/lsgrid/homer/zap.tar
```

- Copy file from local machine to DPM:

```
$globus-url-copy \  
$ file:///`pwd`/zap.tar \  
$ gsiftp://gb-se-lumc.lumc.nl:2811/dpm/lumc.nl/home/lsgrid/homer/zap.tar
```

- Recursive upload to dCache:

```
$globus-url-copy -cd -r \  
$ /home/homer/testdir/ \  
$ gsiftp://gridftp.grid.sara.nl:2811/pnfs/grid.sara.nl/data/lsgrid/homer/testdir/  
## replace testdir with your directory
```

- Recursive upload to DPM:

```
$globus-url-copy -cd -r \  
$ /home/homer/testdir/ \  
$ gsiftp://gb-se-lumc.lumc.nl:2811/dpm/lumc.nl/home/lsgrid/homer/testdir/
```

- Recursive download from dCache:

First create the directory locally, e.g. testdir.

```
$globus-url-copy -cd -r \  
$ gsiftp:///gridftp.grid.sara.nl:2811/pnfs/grid.sara.nl/data/lsgrid/homer/testdir/ \  
$ /home/homer/testdir/
```

- Recursive download from DPM:

First create the directory locally, e.g. testdir.

```
$globus-url-copy -cd -r \  
$ gsiftp://gb-se-lumc.lumc.nl:2811/dpm/lumc.nl/home/lsgrid/homer/testdir/ \  
$ /home/homer/testdir/
```

- Third party transfer (between dCache sites):

First create the remote directory, e.g. targetdir.

```
$globus-url-copy -cd -r \  
$ gsiftp://gridftp.grid.sara.nl:2811/pnfs/grid.sara.nl/data/lsgrid/homer/sourcetdir/ \  
$ gsiftp://gridftp.grid.sara.nl:2811/pnfs/grid.sara.nl/data/lsgrid/penelope/targetdir/  
## note: you must include the trailing slash!
```

See also:

For dCache 3rd party transfers see also *fts client*.

- Third party transfer (between DPM sites):

First create the remote directory, e.g. targetdir.

```
$globus-url-copy -cd -r \
$   gsiftp://gb-se-lumc.lumc.nl:2811/dpm/lumc.nl/home/lsgrid/homer/testdir/ \
$   gsiftp://gb-se-ams.els.sara.nl/dpm/els.sara.nl:2811/home/lsgrid/penelope/testdir/
## note: you must include the trailing slash!
```

Parallel streams The `globus-url-copy` uses by default 10 parallel streams for transfers.

Removing data The `globus-*` client does not offer an option to delete files or directories. For this purpose, use a different client, e.g. *uberftp client*.

srm client

This page explains the use of the `srm` client. For an overview of storage clients, see *Storage clients*.

Contents

- *srm client*
 - *SRM*
 - * *Creating/listing*
 - * *Transferring data*
 - *Recursive transfer*
 - *Parallel streams*
 - * *Removing data*
 - *Recursive delete*

SRM basics

See also:

Have a look at our mooc video *Storage Resource Manager* for additional examples.

SRM

The Storage Resource Manager (short SRM) has been designed to be a single interface for the management of both disk and tape storage resources. It provides options for copying files to/from the Grid storage, *Staging files* from tape, creating or removing files and so on. It uses *SURLs* as the physical filename to reference a file.

The *srm client* is one of the most popular *Storage clients*. However, `srm`- commands are using Java which has the tendency to allocate big amounts of memory and sometimes be slow, also because of the SRM protocol overhead. If transfer speed is important, use *uberftp client*, *globus client* or *gfal client* instead.

Note: To run the examples below you need to have a valid proxy, see *StartGridSession*.

Creating/listing

- Listing directories on dCache:

```
$srmls srm://srm.grid.sara.nl:8443/pnfs/grid.sara.nl/data/lsgrid/
```

- Listing directories on DPM:

```
$srmls srm://gb-se-lumc.lumc.nl:8446/dpm/lumc.nl/home/lsgrid/
```

- Create a new directory on dCache:

```
$srmmkdir srm://srm.grid.sara.nl:8443/pnfs/grid.sara.nl/data/lsgrid/homer/newdir/
```

- Create a new directory on DPM:

```
$srmmkdir srm://gb-se-lumc.lumc.nl:8446/dpm/lumc.nl/home/lsgrid/homer/newdir
```

Transferring data

Note: The `-debug` option would show you extra logging information for your transfers.

- Copy file from dCache to local machine:

```
## note the flag -server_mode=passive!
$srmcpc -server_mode=passive \
$      srm://srm.grid.sara.nl:8443/pnfs/grid.sara.nl/data/lsgrid/homer/zap.tar \
$      file:///`pwd`/zap.tar
```

- Copy file from DPM to local machine:

```
## note the flag -server_mode=passive!
$srmcpc -server_mode=passive \
$      srm://gb-se-lumc.lumc.nl:8446/dpm/lumc.nl/home/lsgrid/homer/zap.tar \
$      file:///`pwd`/zap.tar
```

- Copy file from local machine to dCache:

```
$srmcpc -debug file:///`pwd`/zap.tar \
$      srm://srm.grid.sara.nl:8443/pnfs/grid.sara.nl/data/lsgrid/homer/zap.tar
```

- Copy file from local machine to DPM:

```
$srmcpc -debug file:///`pwd`/zap.tar \
$      srm://gb-se-lumc.lumc.nl:8446/dpm/lumc.nl/home/lsgrid/homer/zap.tar
```

Recursive transfer Recursive transfer of files is not supported with the `srm-*` client commands.

Parallel streams Information not available yet.

Removing data

- Remove a file from dCache:

```
$srmm rm srm://srm.grid.sara.nl:8443/pnfs/grid.sara.nl/data/lsgrid/homer/zap.tar
```

- Remove a file from DPM:

```
$srmm rm srm://gb-se-lumc.lumc.nl:8446/dpm/lumc.nl/home/lsgrid/homer/zap.tar
```

Recursive delete Recursive deletion of files is not supported with the `srm-*` client commands. It is possible to remove a directory as long as it is empty, i.e. content files have been removed.

gfal client

This page includes the basic commands to use `gfal`. For an overview of storage clients, see *Storage clients*.

Contents

- *gfal client*
 - *gFAL*
 - * *Creating/listing*
 - * *Transferring data*
 - *Recursive transfer*
 - *Parallel streams*
 - * *Removing data*

gFAL

Note: To run the examples below you need to have a valid proxy, see *StartGridSession*.

Mandatory environment settings:

```
$export LCG_GFAL_INFOSYS=bdii.grid.sara.nl:2170
```

Note: The examples below will work both with *TURLs* and *SURLs*.

Creating/listing

- Listing directories on dCache:

```
$gfal-ls -l gsiftp://gridftp.grid.sara.nl:2811/pnfs/grid.sara.nl/data/lsgrid/
```

- Listing directories on DPM:

```
$gfal-ls -l gsiftp://gb-se-lumc.lumc.nl:2811/dpm/lumc.nl/home/lsgrid
```

- Create a new directory on dCache:

```
$gfal-mkdir gsiftp://gridftp.grid.sara.nl:2811/pnfs/grid.sara.nl/data/lsgrid/homer/newdir/
```

- Create a new directory on DPM:

```
$gfal-mkdir gsiftp://gb-se-lumc.lumc.nl:2811/dpm/lumc.nl/home/lsgrid/homer/newdir/
```

Transferring data

- Copy file from dCache to local machine:

```
$gfal-copy gsiftp://gridftp.grid.sara.nl:2811/pnfs/grid.sara.nl/data/lsgrid/homer/zap.tar file:
```

- Copy file from DPM to local machine:

```
$gfal-copy gsiftp://gb-se-lumc.lumc.nl:2811/dpm/lumc.nl/home/lsguid/homer/zap.tar file:///`pwd`/
```

- Copy file from local machine to dCache:

```
$gfal-copy file:///`pwd`/zap.tar gsiftp://gridftp.grid.sara.nl:2811/pnfs/grid.sara.nl/data/lsgri
```

- Copy file from local machine to DPM:

```
$gfal-copy file:///`pwd`/zap.tar gsiftp://gb-se-lumc.lumc.nl:2811/dpm/lumc.nl/home/lsguid/homer/
```

Recursive transfer Recursive transfer of files is not supported with the `gfal-copy` command.

Parallel streams Information not available yet.

Removing data

- Remove a file from dCache:

```
$gfal-rm gsiftp://gridftp.grid.sara.nl:2811/pnfs/grid.sara.nl/data/lsguid/homer/zap.tar
```

- Remove a file from DPM:

```
$gfal-rm gsiftp://gb-se-lumc.lumc.nl:2811/dpm/lumc.nl/home/lsguid/homer/zap.tar
```

- Remove whole (non-empty) directory with all content from dCache:

```
$gfal-rm -r gsiftp://gridftp.grid.sara.nl:2811/pnfs/grid.sara.nl/data/lsguid/homer/testdir/
```

- Remove whole (non-empty) directory with all content from DPM:

```
$gfal-rm -r gsiftp://gb-se-lumc.lumc.nl:2811/dpm/lumc.nl/home/lsguid/homer/testdir/
```

webdav client

This page includes the basic commands to use the webdav protocol. For an overview of storage clients, see [Storage clients](#).

Contents

- *webdav client*
 - *Webdav*
 - * *Listing*
 - * *Creating directories*
 - * *Transferring data*
 - *Uploading*
 - *Downloading*
 - *Downloading with proxy authentication*
 - * *Renaming*
 - * *Removing data*
 - *Graphical access*

Webdav

The webdav protocol has the following advantages:

- It supports username & password authentication
- It uses the common port 443. Some overly strict firewalls may block outgoing traffic, but port 443 is so common that it is seldom blocked. However, using webdav to bypass firewalls should be seen as a temporary solution; it would be better to open up your institute's firewall to allow access to the dCache subnet.

It also has disadvantages:

- It is not a high performance transfer protocol. If this is important, use GridFTP instead.
- Support by webdav clients varies widely. Some operations (like renaming a file) may not work in certain clients and circumstances. Modifying/overwriting a file, although the webdav protocol supports it, doesn't work on dCache; you'll need to delete the old file and upload the new file instead.

dCache has the following webdav doors:

URL including port	Authentication method	Redirection behaviour
https://webdav.grid.sara.nl:443	Username/password	Redirects on read
https://webdav.grid.sara.nl:2880	Username/password	No redirects
https://webdav.grid.sara.nl:2882	User certificate or proxy	Redirects on read and write

If you don't know which one you should use, choose the first. It has a good load balancing. The second, on port 2880, may be useful for certain webdav clients that don't support redirects, such as `cadaver`. Use the third one only if you need to use webdav with a certificate or proxy.

`webdav.grid.sara.nl` is a DNS round robin that will direct you to a (more or less) random host in a pool of webdav servers.

Note: To run the examples below you need to have a UI (or CUA (SURFsara's Central User Administration)) account that is configured within dCache and authorized to the data you want to access. Contact us if you need assistance with that.

Listing To list directories, you can point a browser like Firefox to <https://webdav.grid.sara.nl/pnfs/grid.sara.nl/data/>. When the browser asks for a username and password, you can provide your Grid UI (or CUA) username and password. When you click on a listed file, it will be downloaded, when you're authorized to do so. When you're not authorized to access a URL, you may see some unexpected behaviour.

You can also use text browsers like `curl` to list directories.

Creating directories To create a directory with `curl`:

```
$ curl --capath /etc/grid-security/certificates/ --fail --user homer \
$      --request MKCOL https://webdav.grid.sara.nl/pnfs/grid.sara.nl/data/lsguid/homer/directory
```

Transferring data

Uploading To copy a file from your local machine to dCache:

```
$curl --capath /etc/grid-security/certificates/ --fail --location --user homer \  
$ --upload-file zap.tar \  
$ https://webdav.grid.sara.nl/pnfs/grid.sara.nl/data/lsgrid/homer/  
$# replace homer with your username, lsgrid with your VO and zap.tar with your local file
```

The command will ask for the password of ‘homer’ on the command line. If you don’t want to type the password each time, specify `--netrc` and store the password in the `.netrc` file in your home dir. Make sure it is not readable by others (`$ chmod 600 .netrc`). See ‘man curl’ for more details.

Note: It is possible to specify the password on the command line like this: `--user homer:password`. However, for security reasons this should be avoided from share systems (like the UI) because it allows other local users to read the password with the `ps` command.

If on your system there are no Grid CA certificates available in `/etc/grid-security/certificates/`, you can install them by following these instructions: <https://dist.eugridpma.info/distribution/igtfl/>, or you can specify `--insecure` to skip certificate checking (not recommended!).

Downloading To copy a file from dCache to your local machine:

```
$curl --capath /etc/grid-security/certificates/ --fail --location --user homer \  
$ https://webdav.grid.sara.nl/pnfs/grid.sara.nl/data/lsgrid/homer/zap.tar \  
$ --output zap.tar
```

Or with `wget`:

```
$wget --user=homer --ask-password --ca-directory=/etc/grid-security/certificates \  
$ https://webdav.grid.sara.nl/pnfs/grid.sara.nl/data/lsgrid/homer/zap.tar
```

Note: `wget` does not support certificate/proxy authentication.

If you don’t have an `/etc/grid-security/certificates` directory, you could specify `--no-check-certificate`, but we don’t recommend this.

Downloading with proxy authentication To download a file while using a proxy to authenticate, you first have to create your proxy, see [StartGridSession](#).

Then use a command like this:

```
$curl --capath /etc/grid-security/certificates/ \  
$ --cert $X509_USER_PROXY --cacert $X509_USER_PROXY \  
$ https://webdav.grid.sara.nl:2882/pnfs/grid.sara.nl/data/lsgrid/homer/zap.tar
```

Note: It is possible that your proxy DN (Distinguished Name) is mapped to another user account than your own CUA user account. If you have permission issues with either username or proxy and not the other, contact us to check the user mapping.

Renaming Curl can rename files if proxy authentication is used.

```
$curl --capath /etc/grid-security/certificates/ --fail --location \  
$ --cert $X509_USER_PROXY --cacert $X509_USER_PROXY \  
$ --request MOVE \  
$ https://webdav.grid.sara.nl:2882/pnfs/grid.sara.nl/data/lsgrid/homer/oldfile \  
$ --header "Destination:https://webdav.grid.sara.nl:2882/pnfs/grid.sara.nl/data/lsgrid/homer/newfile"
```


File properties and locality are not changed. A file that is stored on tape (nearline) will stay on tape, even if it is moved to a directory for disk-only files.

As far as we know, renaming does not work when username/password authentication is used.

Removing data Deleting a file from dCache:

```
$ curl --capath /etc/grid-security/certificates/ --user homer --location \
$      --request DELETE https://webdav.grid.sara.nl/pnfs/grid.sara.nl/data/lsgrid/homer/zap.tar
```

Graphical access

To work with WebDAV on Windows or Mac OS X, you can install **Cyberduck** from here: <https://cyberduck.io/>. Please note that the App store package costs money; the download from the website is free, but will ask for a donation.

- Download the .zip file, open it, and drag the .app file into your Applications folder to install it.
- Open a WebDAV (HTTP/SSL) connection and connect to the server with your UI account username and password:

```
https://webdav.grid.sara.nl/pnfs/grid.sara.nl/data/lsgrid/ # replace lsgrid with your VO
```

The screenshot shows a configuration window for a WebDAV client. The title bar reads 'webdav.grid.sara.nl - WebDAV (HTTPS)'. The main content area has a header 'WebDAV (HTTP/SSL)' with a dropdown arrow. Below this, the 'Nickname' field is 'webdav.grid.sara.nl - WebDAV (HTTPS)'. The 'URL' field is 'https://homer@webd...a.nl/data/users/homer'. The 'Server' field is 'webdav.grid.sara.nl' and the 'Port' is '443'. The 'Username' field is 'homer'. There is an unchecked checkbox for 'Anonymous Login'. A section titled 'More Options' is expanded, showing a 'Path' field with '/pnfs/grid.sara.nl/data/users/homer'. The 'Connect Mode' and 'Encoding' are both set to 'Default'. There is an unchecked checkbox for 'Use Public Key Authentication' with the text 'No private key selected' below it. The 'Download Folder' is set to 'Downloads'. The 'Transfer Files' is set to 'Default'. The 'Web URL' is 'http://webdav.grid.sara.nl'. There is a 'Notes' text area. The 'Timezone' is set to 'UTC'.

webdav.grid.sara.nl - WebDAV (HTTPS)

WebDAV (HTTP/SSL)

Nickname: webdav.grid.sara.nl - WebDAV (HTTPS)

URL: <https://homer@webd...a.nl/data/users/homer>

Server: webdav.grid.sara.nl Port: 443

Username: homer

☐ Anonymous Login

▼ More Options

Path: /pnfs/grid.sara.nl/data/users/homer

Connect Mode: Default

Encoding: Default

☐ Use Public Key Authentication
No private key selected

Download Folder: Downloads

Transfer Files: Default

Web URL: <http://webdav.grid.sara.nl>

Notes:

Timezone: UTC

fts client

This page includes the basic commands to use the FTS (File Transfer Service). For an overview of storage clients, see *Storage clients*.

Contents

- *fts client*
 - *FTS*
 - * *Authentication*
 - * *FTS file transfers*
 - *fts-transfer-submit*
 - *Basic options*
 - *File transfer - TURL to TURL*
 - *File transfer - TURL to TURL*
 - *File transfer - SRMv2 to SRMv2*
 - * *Monitor Status*
 - *Command line*
 - *Web interface*

FTS

FTS3 is a file transfer service for reliable file transfers across sites or else, *third party* transfers. You cannot use it to transfer files from your local machine to dCache and vice-versa. FTS3 service has been running for years. It was developed by Cern for WLCG data transfers up to petabytes per month.

From the user perspective, it allows data movement, retrying if necessary, monitoring and displaying usage statistics, see [FTS3 wiki](#). From the operations view, it optimises resource usage.

It supports all the basic transfer protocols, such as: GridFTP, SRM, Webdav, HTTPS, xroot. It is open source and can be used from command-line clients or WebFTS, the web-interface version, or python bindings.

The FTS client is currently installed on the UI `ui.grid.sara.nl`. It is not available on the [LSG UI](#) machines.

Authentication To use the FTS (File Transfer Service) you need to create a local proxy. Then ‘fts-transfer-submit’ automatically delegates the proxy to the FTS server (default lifetime is 12 hours). When the remaining lifetime of the stored proxy passes under 4 hours, fts-transfer-submit will automatically delegate a new one as long as there is a valid **local proxy**.

FTS file transfers

Note: To run the examples below you need to have a valid local proxy. The `voms-proxy-init` tool can be used to generate a proxy with VOMS (Virtual Organization Membership Service) attributes from the personal certificate. See [Starting a Grid session](#).

fts-transfer-submit This command submits transfer-jobs by specifying the source and destination file location. The file location can be a SURL (Storage URL), TURL (Transport URL) or HTTPS link. The source and destination endpoints are GridFTP or SRM servers. The output of the command is a *unique ID* that can be used for tracing the transfer status.

Basic options

- `-v`: enable verbose information
- `-s`: specify fts server
- `-K`: enable checksum. By default, Adler32 is supported on the SURFsara servers.

File transfer - TURL to TURL

```
$fts-transfer-submit -s https://fts3.grid.sara.nl:8443 \  
$   gsiftp://gridftp.grid.sara.nl:2811/pnfs/grid.sara.nl/data/lsgrid/homer/zap.tar \  
$   gsiftp://gridftp.grid.sara.nl:2811/pnfs/grid.sara.nl/data/lsgrid/penelope/zap.tar
```

File transfer - TURL to TURL

```
$fts-transfer-submit -s https://fts3.grid.sara.nl:8443 \  
$   gsiftp://gridftp.grid.sara.nl:2811/pnfs/grid.sara.nl/data/lsgrid/homer/zap.tar \  
$   gsiftp://gridftp.grid.sara.nl:2811/pnfs/grid.sara.nl/data/lsgrid/penelope/zap.tar
```

File transfer - SRMv2 to SRMv2

```
$fts-transfer-submit -s https://fts3.grid.sara.nl:8443 \  
$   srm://srm.grid.sara.nl:8443/srm/managerv2?SFN=/pnfs/grid.sara.nl/data/lsgrid/zap.tar \  
$   srm://srm.grid.sara.nl:8443/srm/managerv2?SFN=/pnfs/grid.sara.nl/data/lsgrid/penelope/zap.tar
```

Note: Combinations between TURLS, SURLS, HTTPS and SRMv2 are possible.

Monitor Status

Command line The `fts-transfer-submit` command will return instantly an ID for the specific job. This ID can be used to trace the status of the transfer:

```
$fts-transfer-status -s https://fts3.grid.sara.nl:8443 9e665677-76e5-4734-b729-b69e161da99a  
## replace the string with your ID
```

Web interface You can monitor the transfer status and trace the logging information on this page:

<https://fts3.grid.sara.nl:8449/fts3>

At the moment any jobs are visible to anyone under any VO, but this can be closed by our system administrators upon request, just contact us at helpdesk@surfsara.nl.

globusonline client

This page includes the basic commands to use *globusonline*. For an overview of storage clients, see *Storage clients*.

Contents

- *globusonline client*
 - *GlobusOnline*
 - * *About*
 - * *Requirements*
 - *Installation*
 - * *Globus Connect*
 - *Data transfer workflow*
 - * *Activating endpoints*
 - * *Data transfers via the web interface*

GlobusOnline

About Globus provides a web service to schedule file transfers from one storage endpoint to another. Since the service employs GridFTP as transfer protocol either the source or the sink has to be GridFTP-enabled. The Globus Online web service allows to monitor the transfer processes via the web interface and reschedules transfers automatically when they fail. Storage endpoints can be different servers or only different locations on the same server. The data transfer is executed on behalf of the user employing his/her credentials or Grid certificate.

There exists a python API which allows to steer and monitor the processes from within e.g. the data generating processes. Via this API data transfers can be easily integrated into workflows.

The service is run on the Amazon cloud located in the US.

Requirements Globus online makes use of Grid proxies. Thus, a user needs to have:

- *A personal Grid certificate*
- *A VO membership*

Installation

Globus Connect

- Create an account here: <https://www.globus.org/> by following the instructions.
- To transfer data one needs to define the endpoints. On servers or computers that do not run GridFTP you will first have to install the Globus Connect client software. Below we show an example on how to define your home folder on a Grid User Interface machine as endpoint:
 - Click on the button “Endpoints” followed by “add Globus Connect personal”
 - Provide a name for your endpoint e.g. ui.grid.sara.nl and press “Generate Setup Key”
 - Copy the “Setup Key” to your clipboard
 - For a linux machine as the UI, download the linux package for Globus Connect Personal
 - Copy the software to your home directory on the Grid user interface and unpack:

```
laptop$ scp Downloads/globusconnectpersonal-latest.tgz homer@ui.grid.sara.nl:
UI$ tar -xzf globusconnectpersonal-latest.tgz
```

- Register the new endpoint by running:

```
cd globusconnectpersonal-*
./globusconnect -setup fd409d15-297d-4jl6-8wea-47bjs6b2bc88 # replace fd409d15-297d-4jl6-8wea-47bjs6b2bc88
# Configuration directory: /home/homer/.globusonline/lta
# Contacting relay.globusonline.org:2223
# Installing certificate and key
# Creating /home/homer/.globusonline/lta/gridmap
# Done!
```

- Prior to employing the endpoint the client has to be started:

```
./globusconnect -start &
```

Note: These steps work for any personal (non GridFTP-enabled) server.

- Define other endpoints:

By clicking the button “Endpoints” you can search in a list with all endpoints, e.g. search for `surfsara#dCache`. This GridFTP endpoint points at the *dCache* Grid storage.

- To register your own GridFTP-enabled endpoint click on ” add Globus Connect Server endpoint” and follow the instructions there. You will need to provide MyProxy and GridFTP URLs. In case of the SURFsara Grid you will need to set `MyProxy = px.grid.sara.nl` and `GridFTP = gridftp.grid.sara.nl`

Note: Before data can be transferred you need to register endpoints with Globus Online. This also holds true when working with another client than the web interface.

Data transfer workflow

Activating endpoints Globus Online executes data transfers on behalf of a user. To this end it employs a Grid proxy that is provided by the user and instantiated with his/her Grid credentials/certificates. Independent from using the python API or the web interface one first has to activate the endpoints from/to which data should be transferred.

- The non GridFTP-enabled endpoints like personal workstations or the home of the Grid user interface machines are activated by running:

```
./globusconnect -start &
```

- To activate a GridFTP-enabled endpoint the user needs to provide the service with a Grid proxy. Start a Grid session and create a Grid proxy on the proxy server:


```
startGridSession lsgrid # replace lsgrid with your VO  
myproxy-init --voms lsgrid -l homer # replace lsgrid with your VO and homer e.g. with your name.
```

After that you are asked to authenticate with your Grid certificate password and give a passphrase that will be used afterwards to export the proxy via the web interface.

- Go to the webinterface and click “activate” the Gridftp endpoint. Provide the username and passphrase from the previous step:

Manage Endpoints

[Endpoint List](#)[+ add Globus Connect Personal endpoint](#)[+ add Globus Connect Server endpoint](#)

 **surfsara#dCache_gridftp**
Public Endpoint

not active
[activate](#)

[Overview](#)[Server](#)[My Shares](#)[Activate](#)

Please authenticate to access this endpoint

Login Server px.grid.sara.nl [change](#)

Username

Password

[advanced](#)

[Authenticate](#)

Data transfers via the web interface Data transfers can be easily started employing the web interface. You have to provide the names of the endpoints from and to which the data is transferred:

[Transfer Files](#) | [Activity](#) | [Endpoints](#) | [Bookmarks](#) | [Console](#)

Transfer Files

RECENT ACTIVITY 0 0 0 0

Endpoint: ☆

Path: Go

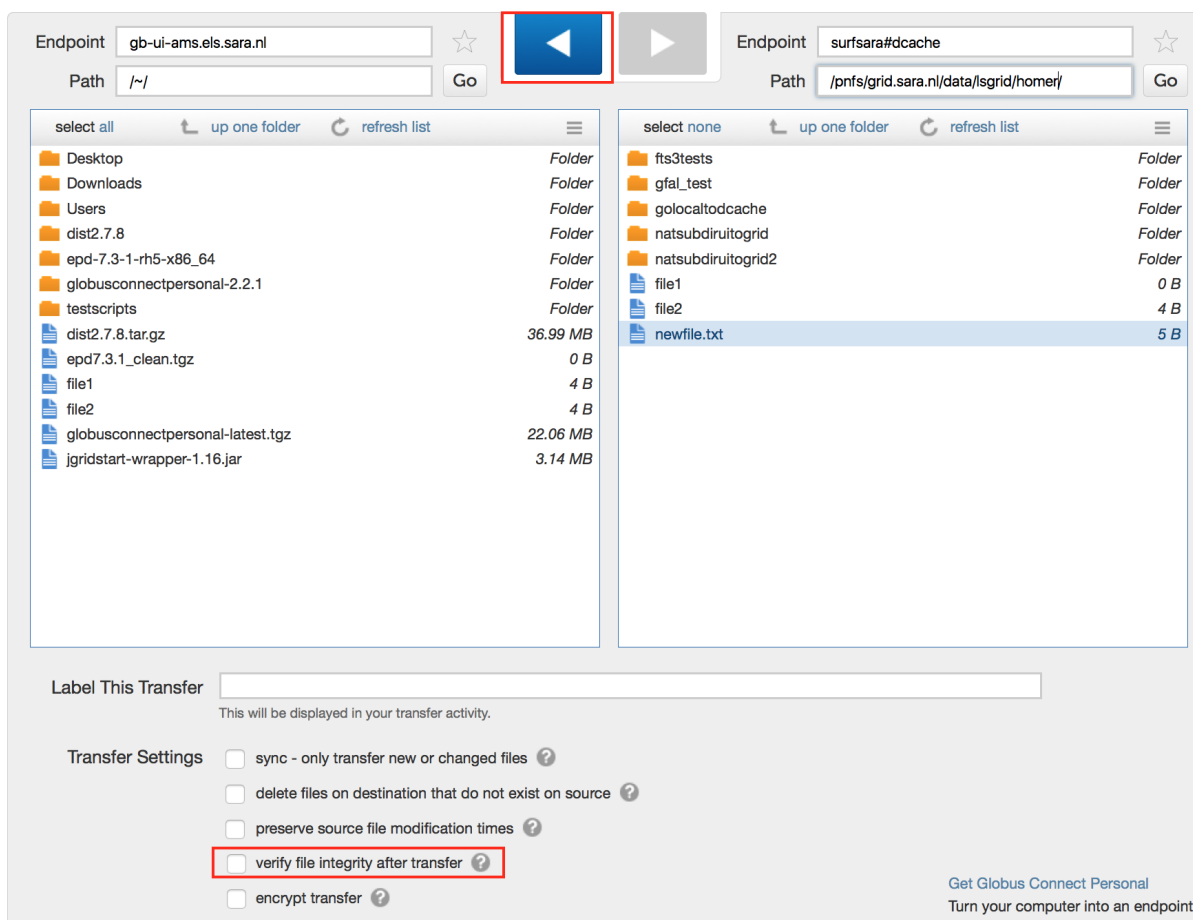
Desktop	Folder
Downloads	Folder
Users	Folder
dist2.7.8	Folder
epd-7.3-1-rh5-x86_64	Folder
globusconnectpersonal-2.2.1	Folder
testscripts	Folder
dist2.7.8.tar.gz	36.99 MB
epd7.3.1_clean.tgz	0 B
file1	4 B
file2	4 B
globusconnectpersonal-latest.tgz	22.06 MB
jgridstart-wrapper-1.16.jar	3.14 MB

Endpoint: ☆
 Path: Go

fts3tests	Folder
gfal_test	Folder
golocaltodcache	Folder
natsubdiruitogrid	Folder
natsubdiruitogrid2	Folder
file1	0 B
newfile.txt	5 B

Data to be transferred is selected by marking it and then clicking one of the arrows to determine sink and source. The current state of data transfers can be monitored in the "Activity" screen:

Warning: To enable transfers from dCache to your personal endpoint, you need to "untick" the box that verifies data integrity due to incompatible checksum methods between Globusonline and our dCache service.



GlobusOnline is an easy graphic-based way to interact with our Grid Storage, but keep in mind that recursive directory transfers can be slow and checksum verification has to be disabled in our current implementation.

lcg-lfn-lfc clients

This page includes the basic commands to use `lcg-lfn-lfc`. For an overview of storage clients, see [Storage clients](#).

Contents

- *lcg-lfn-lfc* clients
 - *About*
 - *lcg* tools
 - * *Creating/listing*
 - * *Transferring data*
 - *Parallel streams*
 - * *Removing data*
 - *Putting lcg-lfn-lfn together*
 - * *Creating/listing*
 - *Replicating files*
 - * *Copying files and registering files in the logical file catalog*
 - *Troubleshooting LFC entries*

Warning: In general we don't recommend the `lcg-lfn-lfc` clients as many users reported difficulties in usage and encountered worse performance compared to the other *Storage clients*. If your solution still works with the `lcg-` tools you can keep on using those; though the `lcg-` tools are *not* supported anymore.

About

The Logical File Catalog (LFC) allows you to give more descriptive names to your files, and to order your files in a directory structure. Bear in mind that the LFC (Logical File Catalog) is not itself a storage system. It's just a database that keeps track of your files.

You can manipulate your data with a set of command line tools. Some of these commands start with `lfc-`, while others start with `lcg-`, which can be confusing at first.

lcg-... All these commands operate on the data itself. Additionally, some of these commands have “side effects” in the LFC: e.g. the `lcg-cr` command uploads a file to an SE and registers this file in the LFC under a certain name.

lfc-... These commands only operate on the LFC, and don't manipulate data. E.g. the command `lfc-mkdir` creates a new directory in the LFC.

lfn-... The `lfn` commands allow you interact with the LFC logical names.

Note: To run the examples below you need to have a valid proxy, see [StartGridSession](#).

lcg tools

Creating/listing

- List dCache directories:

```
$lcg-ls srm://srm.grid.sara.nl:8443/pnfs/grid.sara.nl/data/lsgrid/
```

- List DPM directories:

```
$lcg-ls srm://gb-se-lumc.lumc.nl:8446/dpm/lumc.nl/home/lsgrid/
```

Transferring data

- Copy file from local machine to dCache:

```
$lcg-cp file:`pwd`/zap.tar srm://srm.grid.sara.nl:8443/pnfs/grid.sara.nl/data/lsgrid/homer/zap.t
```

- Copy file from local machine to DPM:

```
$lcg-cp file:`pwd`/zap.tar srm://gb-se-lumc.lumc.nl:8446/dpm/lumc.nl/home/lsgrid/homer/zap.tar
```

Parallel streams Information not available yet.

Removing data

- Remove a file from dCache:

```
$lfcg-del -l srm://srm.grid.sara.nl:8443/pnfs/grid.sara.nl/data/lsgrid/homer/testfile
```

Putting lfc-lfn-lfn together

Creating/listing For each of the supported VOs, a separate “top level” directory exists under the `/grid/` directory. E.g. to see all the files that are stored for the `lsgrid` VO, make sure you have a running `lsgrid` VOMS proxy and then type:

```
$lfc-ls -l /grid/lsgrid/
drwxrwxr-x  2 30125  3010          0 Feb 05 12:56 arni
drwxrwxr-x  3 30146  3010          0 Mar 06 15:21 dutilh
drwxrwxr-x  3 30147  3010          0 Feb 22 16:12 emc-gwatest
...
...
...
```

Rather than having to type an absolute path for every file and directory you use, it is instead possible to define a home directory from which you may use relative file/directory paths. You can do this by setting the environment variable `LFC_HOME`:

```
$export LFC_HOME='/grid/lsgrid'
```

- Creating a new directory:

Before you can register any file of your own, you must create a new directory in the file catalog:

```
$lfc-mkdir /grid/your_vo/your_username
```

- To check that you have created your directory type:

```
$export LFC_HOME=/grid/your_vo
$lfc-ls -l
```

and you should see your directory (plus possibly those of others).

Replicating files

File replication means that you copy the same file to multiple storage elements. If you then start a Grid job which uses that file, and the job lands on one of the compute elements of the Life Science Grid, you then use the file which is nearest to the compute element. This reduces the time needed to copy the file, and reduces network traffic.

You can replicate a file and use the replicas with the following steps:

1. Copy your file to one of the storage elements, while registering the file in the *Logical File Catalog*
2. Replicate the file to other storage elements, and register the copies under the same entry in the *Logical File Catalog*
3. In your job description, tell the scheduler where to run jobs by specifying a *data requirement*

This section describes the steps.

Copying files and registering files in the *logical file catalog* To copy a file from a user interface to one of the storage elements, and register the file in the logical file catalog:

- determine the full path of the file; for example, using the `pwd` command:

```
$pwd
/home/homer/Projects/input.dat
```

- determine the full path of the target file, on *dCache* or *DPM*; see *Grid file identifiers* about how to refer to the target file.
- use `lcg-cr` and the full path to the file to store the first copy of your file on one of the Storage Elements, and register the file in the *logical file catalog*:

```
$lcg-cr --vo lsgrid \
$      -d srm://gb-se-kun.els.sara.nl/dpm/els.sara.nl/home/lsgrid/homer/input.dat \
$      -l lfn:/grid/lsgrid/homer/input.dat \
$      file:///home/homer/Projects/input.dat
```

In this example, the file `input.dat` is copied from the `Projects` directory on the local user interface, to a storage element on the Life Science Cluster in Nijmegen, and registered in the LFC, with the credentials from the VO *lsgrid*. Note that this requires membership of the *lsgrid* VO.

- use `lcg-rep` to create a replica of the file, and register the replica with the LFC:

```
$lcg-rep \
$      -d srm://gb-se-amc.amc.nl/dpm/amc.nl/home/lsgrid/homer/input.dat \
$      lfn:/grid/lsgrid/homer/input.dat
```

Note that the LFC location is the same as in the `lcg-cr` command.

- verify that there are two copies of the file, registered under the same LFC entry:

```
$lcg-lr lfn:/grid/lsgrid/homer/input.dat
srm://gb-se-kun.els.sara.nl/dpm/els.sara.nl/home/lsgrid/homer/input.dat
srm://gb-se-amc.amc.nl/dpm/amc.nl/home/lsgrid/homer/input.dat
```

Troubleshooting LFC entries

Note: The LFC needs to support your VO in order to work.

The LFC is a place where you register files, so you can find their replicas that are physically stored on a Storage Element.

If the physical storage is removed or lost, and you don't have any other replica's, you end up with only a registration in the LFC.

```
-----
Setting up a testfile to reproduce the situation:
```

```
Copy and register a testfile.
```

```
lcg-cr testfile -l lfn:/grid/lsgrid/homer/demo/testfile
```

```
Deleting the srm entry and not the lfc entry.
```

```
lcg-del --nolfc srm://srm.grid.sara.nl/pnfs/grid.sara.nl/data/lsgrid/generated/2015-06-05/file25a858
```

```
-----
Trying to delete the lfc entry:
```

```
lcg-del -a lfn:/grid/lsgrid/homer/demo/testfile
```

```
Gives me the error:
```

```
[SE][advisoryDelete] http://srm.grid.sara.nl:8443/srm/managerv1: java.rmi.Remote
Exception: srm advisoryDelete failed; nested exception is:
java.lang.RuntimeException: advisoryDelete(User [name=lsgrid, uid=18050,
```

```
gids=[18050], root=],pnfs/grid.sara.nl/data/lsgrid/generated/2015-06-05/file25a
8581b-1d76-4579-ab1f-5d2e8e58b33c) Error file does not exist, cannot delete
```

To remove the lfc entry you can use a
Lcg-uf [guid] [surl] command:

```
List guid
Lcg-lg lfn://grid/lsgrid/homer/demo/testfile
```

```
List registered replica's SURL(s)
Lcg-lr lfn://grid/lsgrid/homer/demo/testfile
```

Issue unregister command to remove the lfc entry:

```
lcg-uf guid:644ee342-clf8-4964-b878-a4bd5ccb3d6a srm://srm.grid.sara.nl/pnfs/grid.sara.nl/data/lsgrid/
```

Or shorter command doing exactly the same:

```
f=lfn://grid/lsgrid/homer/demo/testfile lcg-uf $(lcg-lg $f) $(lcg-lr $f)
```

3.2.5 Staging files

The *dCache* storage at SURFsara consists of magnetic tape storage and hard disk storage. If your *quota allocation* includes tape storage, then the data stored on magnetic tape has to be copied to a hard drive before it can be used. This action is called *Staging files* or ‘bringing a file online’.

Table 3.2: Staging terms

Locality	Meaning
ONLINE	The file is only on disk
NEARLINE	The file is only on tape; it should be staged before reading it
ONLINE_AND_NEARLINE	The file is both on disk and on tape

Staging a single file

Note: For all staging operations you need to have a valid proxy, see *StartGridSession*.

Here is an example of how to stage a single file:

```
$srm-bring-online srm://srm.grid.sara.nl/pnfs/grid.sara.nl/data/lsgrid/test
srm://srm.grid.sara.nl/pnfs/grid.sara.nl/data/lsgrid/test brought online, use request id 424966221 t
```

Don’t use this method to stage multiple files. Use the `stage.py` example below instead, because it is much more efficient.

How to display the locality:

```
$srm ls -l srm://srm.grid.sara.nl/pnfs/grid.sara.nl/data/lsgrid/test | grep locality
locality:ONLINE_AND_NEARLINE
```

Staging groups of files

The example below shows how to stage a list of files with known SURLs.

- Copy and untar the tarball staging scripts to your UI home directory.
- Create a proxy on the UI:

```
$startGridSession lsgrid
```

- The file paths should be listed in a file called `files` with the following format:

```
/pnfs/grid.sara.nl/data/...
```

Let's say that you have a list of SURLS that you want to stage. Convert the list of SURLS in the `datasets/example.txt` file to the desired `/pnfs` format:

```
$grep --only-matching '/pnfs/grid.sara.nl.*' datasets/example.txt > files
```

- Display the locality of the files with:

```
$python state.py
```

- Stage the files:

```
$python stage.py
```

This script stages a number of files from tape. You can change the pin lifetime in the `stage.py` script by changing the `srmv2_desiredpintime` attribute in seconds.

Monitor staging activity

Once you submit your stage requests, you can use the `gfal` scripts to monitor the status or check the webpage below that lists all the current staging requests:

<http://dcmain.grid.sara.nl:2288/poolInfo/restoreHandler/lazy>

Unpin a file

Your files may remain *online* as long as there is free space on the disk pools. When a pool group is full and free space is needed, dCache will purge the least recently used cached files. The tape replica will remain on tape.

The disk pool where your files are staged has limited capacity and is only meant for data that a user wants to process on a Grid site. When you pin a file you set a *pin lifetime*. The file will not be purged until the pin lifetime has expired. Then the data may be purged from disk, as soon as the space is required for new stage requests. When the disk copy has been purged, it has to be staged again in order to be processed on a Worker Node.

When a pool group is full with pinned files, staging is paused. Stage requests will just wait until pin lifetimes for other files expire. dCache will then use the released space to stage more files until the pool group is full again. When this takes too long, stage requests will time out. So pinning should be used moderately.

When you are done with your processing, we recommend you release (or unpin) all the files that you don't need any more. In order to unpin a file, run from the UI:

```
$srm-release-files srm://srm.grid.sara.nl:8443/pnfs/grid.sara.nl/data/lsgrid/homer/zap.tar # replace
```

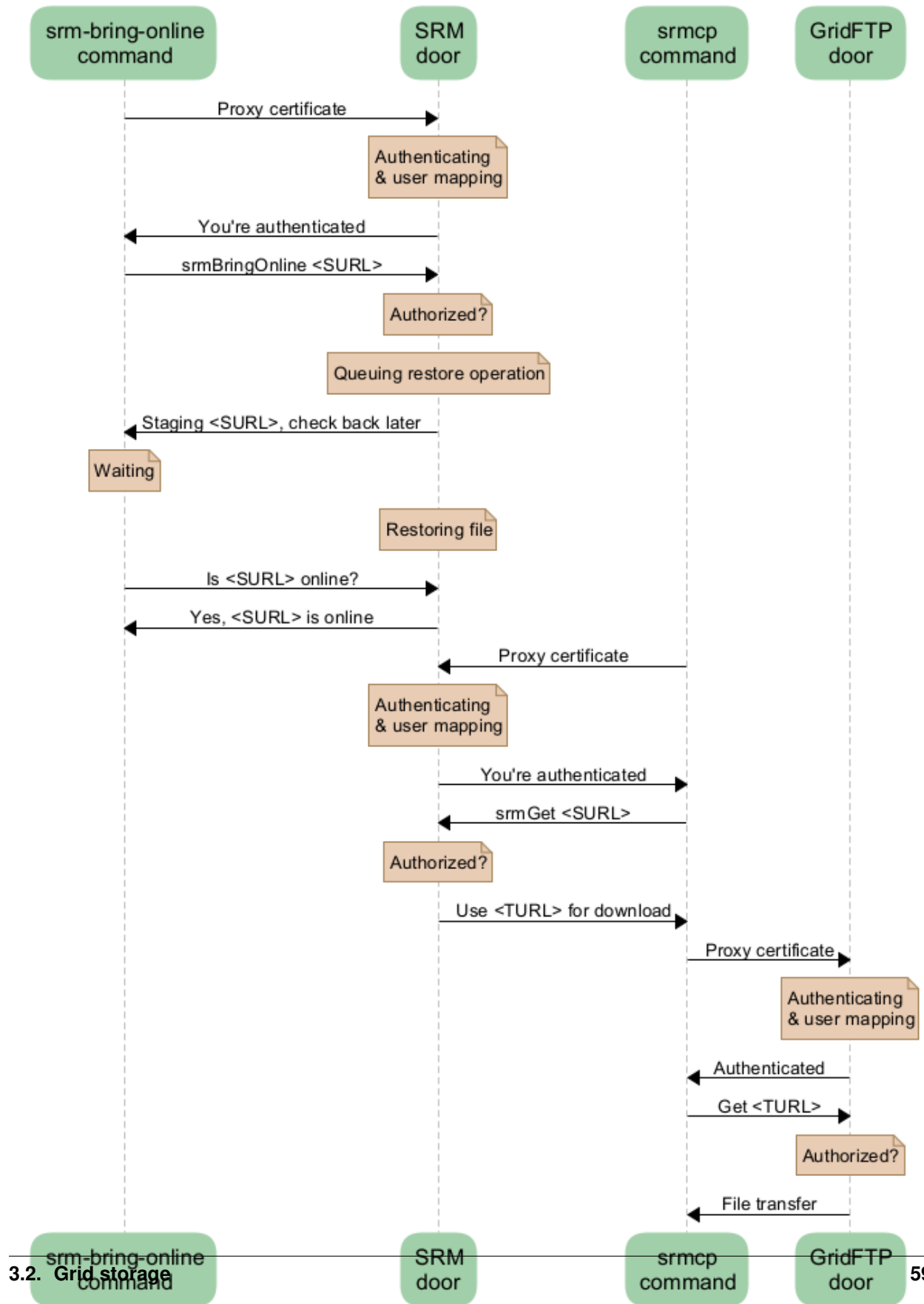
This command will initiate unpinning of file `zap.tar` (even if you submitted multiple pin requests) and the file will remain cached but purgeable until new requests will claim the available space. It is an optional action, but helps a lot with the effective system usage.

Warning: At the moment neither the `srm-bring-online` nor the python `gfal` scripts can effectively release a file if there are multiple pin requests. Please use `srm-release-files`.

3.2.6 SRM interaction example diagram

Here is a sequence diagram that illustrates how the SRM commands interact with the Grid storage.

Using SRM



As you can see from this diagram, there can be a lot of overhead per file. For this reason, Grid storage performs best with large files. We recommend to store files of several megabytes to gigabytes. Some users have files of more than 2 terabytes, but that may be impractical on other systems with limited space like worker nodes.

3.2.7 Importing large amounts of data

The [Data Ingest Service](#) is a SURFsara service for researchers who want to store or analyze large amounts of data at SURFsara. The service is convenient for users who lack sufficient bandwidth or who have stored their data on a number of external hard disks.

3.3 Grid job requirements

By telling what your job needs, you help the scheduler in finding the right place to run your jobs, and it also helps using the different compute nodes in the Grid efficiently.

This chapter describes how to write the requirements, how the requirements determine where your jobs will run, and what they tell the scheduler.

Contents

- *Grid job requirements*
 - *Requirement syntax*
 - *Requirements*
 - * *Specifying Wall Clock time*
 - * *Selecting particular compute elements*
 - * *Multicore jobs*
 - * *Requesting a cluster with a minimum number of cores per node*

3.3.1 Requirement syntax

Job requirements are written as an optional statement in the JDL file:

```
Requirements = <expression>;
```

Job requirements follow the JDL syntax. This also means that you can have multiple requirements using boolean operators `&&` for *requirement 1 AND requirement 2*, and `||` for *requirement 1 OR requirement 2*. You can also use parentheses `(. . .)` for an even more fine-grained control over requirements.

See also:

For detailed information about JDL attributes supported by the gLite Workload Management System, have a look in the [EGEE JDL guide](#).

3.3.2 Requirements

Specifying Wall Clock time

Parameter: `other.GlueCEPolicyMaxWallClockTime`

Synopsis:


```
# make sure that the job can run in a long queue of 72 hours (72 x 60 = 4320 minutes)
Requirements = other.GlueCEPolicyMaxWallClockTime >= 4320;
```

By specifying the wall clock time requirement, the scheduler picks a queue which is long enough for running the job. The parameter is `other.GlueCEPolicyMaxWallClockTime`, the value is **in minutes**. Make sure that your requirement uses the ‘greater than or equal to’ syntax (`>=`).

Jobs in short queues tend to get a higher priority, jobs in long queues tend to get a lower priority. You can use the [queues guideline](#) for determining in which queue your job will run. Note that you need to convert the hours in minutes in your JDL requirement, e.g.:

queue	job length in minutes
short	240 (= 4 hours)
medium	2160 (= 36 hours)
long	4320 (= 72 hours)

Selecting particular compute elements

Parameter: `other.GlueCEInfoHostName`, `other.GlueCEUniqueID`

Synopsis:

```
# Only run on the AMC cluster
Requirements = (
  other.GlueCEInfoHostName == "gb-ce-amc.amc.nl"
);

# Run on the WUR or on the LUMC cluster
Requirements = (
  other.GlueCEInfoHostName == "gb-ce-amc.amc.nl"      ||
  other.GlueCEInfoHostName == "gb-ce-lumc.lumc.nl"
);

# Avoid one of SURFsara's Gina compute elements
Requirements = (other.GlueCEInfoHostName != "creamce2.gina.sara.nl");

# Exclude a specific site, e.g. iihe.ac.be
Requirements=(!RegExp("iihe.ac.be", other.GlueCEUniqueID));

# Schedule the jobs on a specific site, e.g. Gina
Requirements=(RegExp("gina", other.GlueCEUniqueID));
```

With the `other.GlueCEInfoHostName` criterion you can specify on which compute element your jobs will be scheduled. Or even on which CE your jobs will *not* be scheduled. This is convenient in cases where you know jobs will fail on particular systems, for some reason.

`other.GlueCEInfoHostName` contains the hostname, while `other.GlueCEUniqueID` contains the full CE endpoint name including the queue. You can lookup these with the command `lcg-infosites --vo lsgrid ce` (see [example](#)). The last field is the `GlueCEUniqueID`.

Multicore jobs

Parameters: `SmpGranularity`, `CPUNumber`

Synopsis:

```
# Request just 4 cores on a single node
SmpGranularity = 4;
CPUNumber = 4;
```

CPUNumber is the number of cores requested. SMPGranularity is the number of cores that must be scheduled on the same host.

Note that if you do not specify SmpGranularity the requested number of cores (CPUNumber) can be distributed over different nodes, which is only useful for MPI (or likewise) applications.

Warning: If you are running a multi-core process in your job, and you do not set the correct number of CPU cores, **you will oversubscribe a compute node, slowing down your own analysis, as well as others.**

Requesting a cluster with a minimum number of cores per node

Parameter: other.GlueHostArchitectureSMPSize

Synopsis:

```
# request a machine with at least 6 cpu cores on one node
Requirements = (other.GlueHostArchitectureSMPSize >= 6);

# job uses 4 cores
CPUNumber = 4;
SMPGranularity = 4;
```

The default is to select a cluster with GlueHostArchitectureSMPSize >= SmpGranularity. For efficient job allocation on a cluster it is often better to request a number of cores which is less than the GlueHostArchitectureSMPSize (i.e. the number of cores per node).

3.4 Grid authentication

This section explains the concepts and operations regarding Grid authentication mechanisms:

Contents

- *Grid authentication*
 - *Introduction: delegation of authentication*
 - *Starting a Grid session*
 - * *Using VOMS proxies*
 - *Creating a VOMS proxy*
 - *Inspecting your proxy certificate*
 - * *Using the MyProxy server*
 - *Inspecting the myproxy certificate*
 - * *Credential delegation*
 - *Commands for viewing your proxy information*

3.4.1 Introduction: delegation of authentication

Grid, by its very nature, is decentralized. This means that users must authenticate themselves to the Grid services they want to use. This is accomplished by means of a personal certificate and accompanying private key that every Grid user

must have. The combination of a certificate and private key uniquely identifies a user. Therefore, you should **never share your private key** with anyone else or with any service. At the same time your jobs will typically run on systems you may not trust. However, to be able to use those systems you must identify yourself with those systems. This is where *delegation* comes in: identifying yourself with a system you don't trust by creating a new certificate/private key pair, called a proxy, with a limited validity. This chapter describes how you can delegate your credentials.

The easiest way is to use a *Grid session*, which does everything for you in one go.

3.4.2 Starting a Grid session

The easiest way to start a session on the Grid is to use the `startGridSession <VO Name>` command (see also [here](#)) on a user interface (UI) machine.

More about creating proxies?

See also:

For more detailed information about the proxies, have a look at our mooc video [Start a Grid_Session](#).

The `startGridSession` command:

- generates a *local proxy* of your certificate and private key;
- delegates this local proxy to the *Myproxy server*;
- delegates this local proxy to the WMS with your user name as the *delegation ID* (DID).

Your jobs will now be able to run for week. The WMS that is responsible for scheduling your jobs, will renew the proxy certificate of running jobs *every 12 hours* automatically, for one week. This means that your jobs must finish within a week from starting the Grid session. However, running the command again will extend your jobs with a week of run time.

Note: Every time you submit the `startGridSession` command it renews your Grid session for an additional week.

Instead of `startGridSession`, you can run the following three commands separately with the same results:

```
# 1. VOMS server: create a voms proxy with voms extensions that enables you to access the Grid for *.
voms-proxy-init --voms lsgrid #replace lsgrid with your VO

# 2. MyProxy server: store a *week* long proxy certificate in the Myproxy server; useful for jobs th
# running for more than 12 hours.
myproxy-init -d -n

# 3. WMS: delegate your credentials to the Workload Management System.
glite-wms-job-delegate-proxy -d $USER
```

The next section explains the `startGridSession` operations step-by-step. See also `startGridSession -h`.

Using VOMS proxies

In order to use the Grid facilities, you have to create a proxy. A proxy is a short-lived certificate/private key combination which is used to perform actions on the Grid on your behalf without using passwords. You can read more in this [paper](#) from Globus Alliance Members. Proxies contain both a certificate and private key and, therefore, should never leave the system. Instead, proxies are delegated to other systems: a new proxy is created on a remote system using the local proxy as authentication. Services that have been provided with a delegation of your proxy can act on your behalf.

The proxy file on the UI is owned by you and placed in the `/tmp` directory. You only deal with this file directly in exceptional cases.

Creating a VOMS proxy

Make sure you have installed your certificate and private on the Grid user interface that you are working on. They should be placed in the `.globus` directory under your home directory and should be named `usercert.pem` and `userkey.pem`. They must have the following ownerships and permissions:

```
$ls -l $HOME/.globus/usercert.pem
-rw-r--r-- 1 homer homer 1956 Nov 16 12:20 /home/homer/.globus/usercert.pem

$ls -l $HOME/.globus/userkey.pem
-r----- 1 homer homer 1956 Nov 16 12:20 /home/homer/.globus/usercert.pem
```

where `homer` should be replaced with your username.

Now issue the following command to create a *local* proxy. The pass phrase you are asked for, is your Grid certificate password:

```
$voms-proxy-init --voms lsgrid
```

You will see the following output in your terminal:

```
Enter GRID pass phrase for this identity:
Contacting voms.grid.sara.nl:30018 [/O=dutchgrid/O=hosts/OU=sara.nl/CN=voms.grid.sara.nl] "lsgrid".
Remote VOMS server contacted successfully.
Created proxy in /tmp/x509up_u39111.
Your proxy is valid until Thu Jan 05 02:07:29 CET 2016
```

This proxy is your “username” for the Grid. The last line in the example shows the expiration time of the proxy.

Non standard location To store your local proxy in a non standard location, use the `-out` option:

```
$voms-proxy-init -voms lsgrid --valid 168:00 -out /home/homer/my_proxy_cert
```

See `voms-proxy-init -h` for more options.

Inspecting your proxy certificate

You can inspect your local proxy with the command:

```
$voms-proxy-info -all
```

Here is an example:

```
subject   : /O=dutchgrid/O=users/O=sara/CN=Homer Simpson/CN=proxy
issuer    : /O=dutchgrid/O=users/O=sara/CN=Homer Simpson
identity  : /O=dutchgrid/O=users/O=sara/CN=Homer Simpson
type      : full legacy globus proxy
strength  : 1024
path      : /tmp/x509up_u39111
timeleft  : 11:48:24
key usage : Digital Signature, Key Encipherment, Data Encipherment
=== VO lsgrid extension information ===
VO        : lsgrid
subject   : /O=dutchgrid/O=users/O=sara/CN=Homer Simpson
```

```

issuer      : /O=dutchgrid/O=hosts/OU=sara.nl/CN=voms.grid.sara.nl
attribute   : /lsgrid/Role=NULL/Capability=NULL
attribute   : /lsgrid/SARA/Role=NULL/Capability=NULL
timeleft    : 11:48:24

```

You can see that a proxy certificate has a limited lifetime and is stored in the `/tmp` directory. VO extension information is also shown and is used to verify if you are indeed a member of this VO and group: A Grid service that has been provided with a delegation of your proxy can contact the VOMS service for membership information and subsequently grant or deny you access.

Note: In the *next step*, you will delegate your proxy certificate to the proxy server and there it will be valid by default for a week. So it will be possible for long running jobs and jobs that started running only after a few days to continue to run. However, the proxy certificate that you use locally is only valid for 12 hours. So remember that after 12 hours you have to create a new proxy certificate to interact with the Grid (and your long running jobs).

Using the MyProxy server

The following command stores a proxy certificate in the proxy server where it will issue new proxy certificates on your behalf for a week. This is necessary for jobs that need more than 12 hours to run.

Issue this command on the UI:

```
$myproxy-init -d -n
```

You should get something like this:

```

Your identity: /O=dutchgrid/O=users/O=sara/CN=Homer Simpson
Enter GRID pass phrase for this identity:
Creating proxy ..... Done
Proxy Verify OK
Your proxy is valid until: Wed Jan 13 14:25:06 2016
A proxy valid for 168 hours (7.0 days) for user /O=dutchgrid/O=users/O=sara/CN=Homer Simpson now exists

```

The delegated proxy can be received locally from other authorized Grid machines.

Inspecting the *myproxy* certificate

You can inspect the *myproxy* certificate with the command:

```
$myproxy-info -d
```

Here is an example of the displayed output:

```

username: /O=dutchgrid/O=users/O=sara/CN=Homer Simpson
owner: /O=dutchgrid/O=users/O=sara/CN=Homer Simpson
timeleft: 167:56:36 (7.0 days)

```

Credential delegation

This section explains the usage of the command `glite-wms-job-delegate-proxy`, which is also executed when running the *startGridSession*.

When you submit a job to the Grid it will be sent to the Workload Management System (WMS). This system will then schedule your job and send it to a worker node somewhere on the Grid. The job will be run on your behalf, therefore, you should delegate your credentials to the WMS WORKLOAD MANAGEMENT SYSTEM.

Credential delegation solves the following problem: when the Grid is busy or when you submit a large number of jobs, it can take more then the standard 12 hours for the jobs to start than your local proxy certificate is valid. The solution is to use *proxy delegation* before submitting jobs.

We assume that you have issued the `voms-proxy-init` command and have a valid local proxy. If not, please see [voms-proxy-init command](#).

To delegate your proxy to the WMS, run on the UI:

```
$echo $USER
$glite-wms-job-delegate-proxy -d $USER # the $USER is the delegation id
```

The variable `$USER` is the delegation id (in this case your login name from the system). This string is needed in other commands to identify your session. In general, you can use any string you like after the `-d` option.

Instead of creating a delegation ID with `-d`, the `-a` option can be used. This causes a delegated proxy to be established automatically. In this case you do not need to remember a delegation identifier. However, repeated use of this option is not recommended, since it delegates a new proxy each time the commands are issued. Delegation is a time-consuming operation, so it's better to use the `-d $USER` when submitting a large number of jobs one after the other.

Here is an example of the displayed output:

```
Connecting to the service https://wms2.grid.sara.nl:7443/glite_wms_wmproxy_server
===== glite-wms-job-delegate-proxy Success =====

Your proxy has been successfully delegated to the WMPProxy(s):
https://wms2.grid.sara.nl:7443/glite_wms_wmproxy_server
with the delegation identifier: homer

=====
```

3.4.3 Commands for viewing your proxy information

- To start your Grid session:

```
$startGridSession lsgrid # replace lsgrid with your VO
```

- To see how much time there is left on your Grid session:

```
$myproxy-info -d
```

- To renew your Grid session:

```
$startGridSession lsgrid #replace lsgrid with your VO
```

- To end your session:

```
$myproxy-destroy -d
```

- To remove your local `/tmp/x509up_uXXX` proxy:

```
$voms-proxy-destroy
```

Note: `myproxy-destroy` will not terminate any job. Jobs will continue to run and will fail when the the proxy certificate that was used at the time of submission, expires. Use [*glite-wms-job-cancel*](#) to cancel running jobs.

3.5 Grid certificates

In this section we discuss Grid user certificates in more detail; how to convert certificates, and how to find out the details of your Grid certificate.

Contents

- *Grid certificates*
 - *Certificate and key file inspection*
 - * *Using the modulus to see whether a key and a certificate match*
 - * *Finding the expiry date of your certificate*
 - * *Finding the subject of your certificate*
 - *Conversion of key and certificate formats*
 - * *Converting from PKCS12 to PEM*
 - * *Converting from PEM to PKCS12*

Once you have obtained and installed a *legacy certificate* there are some `openssl` commands that can help you inspect the information stored in your certificate. This section will show you how to do this.

3.5.1 Certificate and key file inspection

Sometimes you want to view the details of your key and/or your certificate file. Details like:

- do the key and the certificate file go together?
- when does the certificate expire?
- what is the subject or DN (Distinguished Name) of the certificate?

Using the `openssl` command, you can find out the details of your certificates and keys.

Using the modulus to see whether a key and a certificate match

The modulus is a short message that can be used to identify if a private key and certificate match. If they do not match, the private key and certificate are useless.

To find the modulus of your key, use:

```
openssl rsa -in userkey.pem -noout -modulus | openssl md5
```

You will be asked to provide the password you used to protect your key file. To find the modulus of your certificate, use:

```
openssl x509 -in usercert.pem -noout -modulus | openssl md5
```

If the md5 sum of the moduli of the key file and the certificate file do not match, you cannot use that combination to identify yourself.

Finding the expiry date of your certificate

To find out when your certificate is valid, use:

```
openssl x509 -in usercert.pem -noout -dates
```

This will tell you when your certificate is valid.

Note that a key does not have a validity period.

Finding the subject of your certificate

The subject or DN of a certificate is the human-readable identification of who the certificate belongs to. It usually contains your name, country, organisation and your e-mail address.

To find out who the certificate belongs to, use:

```
openssl x509 -in usercert.pem -noout -subject
```

3.5.2 Conversion of key and certificate formats

Private keys and certificates can be stored in different formats. Different systems use different formats. The two important formats are:

- PEM: stores keys and certificates in separate ascii-files; this format is used by the Grid middleware and storage programs;
- PKCS12: stores keys and certificates in one binary file; this format is used by browsers.

DigiCert creates PKCS12 files, whereas *DutchGrid* creates PEM files.

Converting from PKCS12 to PEM

To convert a PKCS12 file to the PEM format, you need two commands; one to extract the key, and one to extract your certificate.

To extract your key, run:

```
openssl pkcs12 -in browsercert.p12 -out userkey.pem -nocerts
```

Note that you will first need to enter the password that was used to *create* the PKCS12 file. Next, you need to enter a password to protect the exported key. Enter that password again to verify. Note that you must enter a password and the password must be at least 12 characters and include non-alphanumerics; if the password is too short, `openssl` will fail without error. You can use the same password as for the PKCS12 file.

To extract your certificate, run:

```
openssl pkcs12 -in browsercert.p12 -out usercert.pem -nokeys -clcerts
```

Converting from PEM to PKCS12

To convert your certificate in PEM format to the PKCS12-format, use:

```
openssl pkcs12 -export -inkey userkey.pem -in usercert.pem -out browsercert.p12
```


This will ask you for a password three times: the first is to unlock your private key stored in the file `userkey.pem`. The PKCS12-file will be password protected, which needs a new password, and the same password for confirmation. Note that you can use the same password as for the private key, but this is not required.

When you import the PKCS12-file into your browser or keychain, you need to enter the password you used to protect the PKCS12-file.

Local jobs on the Life Science Grid

4.1 Life Science Grid clusters

The current page provides information about the Life Science Grid (LSG) clusters and their locations, local support contact persons, and DNS addresses of User Interface (UI) and Storage Element (SE) machines.

Contents

- *Life Science Grid clusters*
 - *About*
 - *Locations and local support contacts*
 - *Cluster details*
 - *Additional Storage Elements*

If you have any questions concerning access to the clusters below, please contact us at helpdesk@surfsara.nl.

4.1.1 About

The *Life Science Grid* is a compute and data processing infrastructure that is available for all Life Science researchers working in the Netherlands. The Life Science Grid consists of a series of computer clusters that can be used either as a local facility or as an integrated Grid environment. For both types of usage, you will need an account on a User Interface machine. Accounts on the user interface machines are managed via a central system at SURFsara, and can be arranged after verification by a *local site support* contact person. To arrange access, the most efficient way is to ask (one of) your local support contact(s) to send a request to our helpdesk for access to the local User Interface machine. Your local support person will need your name, e-mail address, telephone number and nationality, and will verify that you are employed at the relevant institution and an active researcher in one of the Life Science disciplines.

4.1.2 Locations and local support contacts

Location	LSG Site	Local support contact(s)	Notes
AMC, Amsterdam	LSG_AMC	Silvia Olabarriaga Juan Luis Font	
Delft University of Technology	LSG_TUD	Robbert Eggermont	
UMCG, Groningen	LSG_RUG	Peter Horvatovich Morris Swertz	
LUMC, Leiden	LSG_LUMC	Michèle Huijberts	
Maastricht University	LSG_UM	Maarten Coonen Nuno Nunes	
Radboud University, Nijmegen	LSG_KUN	Christian Gilissen Gert Vriend Janita Bralten	Includes a 2TB RAM High-memory node; contact helpdesk@surfsara.nl for access.
Erasmus MC, Rotterdam	LSG EMC	Henri Vrooman	
Utrecht University	LSG_BCBR	Alexandre Bonvin Johan van der Zwan	
VU University, Amsterdam	LSG_VU	René Pool	
Wageningen University	LSG_WUR	Harm Nijveen	

4.1.3 Cluster details

The following table lists the DNS addresses of the User Interface and Storage Element services, for each of the LSG sites.

LSG Site	User interface	Storage element	LSG Storage SURL
LSG_AMC	ui.lsg.amc.amc.nl	gb-se-amc.amc.nl	lsrcm://gb-se-amc.amc.nl:8446/dpm/amc.nl/home/lsggrid/
LSG_TUD	ui.lsg.tud.ewi.tudelft.nl	gb-se-tud.ewi.tudelft.nl	lsrcm://gb-se-tud.ewi.tudelft.nl:8446/dpm/ewi.tudelft.nl/home/lsggrid/
LSG_RUG	ui.lsg.rug.sara.usor.nl	gb-se-rug.sara.usor.nl	lsrcm://gb-se-rug.sara.usor.nl:8446/dpm/sara.usor.nl/home/lsggrid/
LSG_LUMC	ui.lsg.lumc.lumc.nl	gb-se-lumc.lumc.nl	lsrcm://gb-se-lumc.lumc.nl:8446/dpm/lumc.nl/home/lsggrid/
LSG_UM	ui.lsg.maastrichtuniversity.nl	gb-se-lsg-maastrichtuniversity.nl	lsrcm://se.lsg.maastrichtuniversity.nl:8446/dpm/lsg.maastrichtuniversity.nl/home/lsggrid/
LSG_KUN	ui.lsg.kun.els.sara.nl	gb-se-kun.els.sara.nl	lsrcm://gb-se-kun.els.sara.nl:8446/dpm/els.sara.nl/home/lsggrid/
LSG EMC	ui.lsg.emc.erasmusmc.nl	gb-se-emc.erasmusmc.nl	lsrcm://gb-se-emc.erasmusmc.nl:8446/dpm/erasmusmc.nl/home/lsggrid/
LSG_BCBR	ui.lsg.bcbrr.uu.nl	se.lsg.bcbrr.uu.nl	lsrcm://se.lsg.bcbrr.uu.nl:8446/dpm/lsg.bcbrr.uu.nl/home/lsggrid/
LSG_VU	ui.lsg.psy.vu.nl	se.lsg.psy.vu.nl	lsrcm://se.lsg.psy.vu.nl:8446/dpm/lsg.psy.vu.nl/home/lsggrid/
LSG_WUR	ui.lsg.wur.els.sara.nl	gb-se-wur.els.sara.nl	lsrcm://gb-se-wur.els.sara.nl:8446/dpm/els.sara.nl/home/lsggrid/

User Interface machines can be accessed with any SSH client. To learn how to access the Storage Elements, see [Grid storage](#)

4.1.4 Additional Storage Elements

The following Storage Elements are also available when you are member of the `lsgrid` VO.

Site	Storage element	LSG Storage SURL	Remarks
SURFsara	<code>srm://srm.grid.sara.nl</code>	<code>srm://srm.grid.sara.nl:8443/pnfs/grid.sara.nl/data/lsgrid</code>	See Grid
Target Groningen	<code>srm://srm.target.rug.nl</code>	<code>srm://srm.target.rug.nl:8444/lsgrid</code>	
Test Cluster SURFsara	<code>srm://gb-se-ams.els.sara.nl</code>	<code>srm://gb-se-ams.els.sara.nl:8446/dpm/lsgrid</code>	Do not use for testing purpose only

4.2 PBS local jobs

In this page we will talk about job submission to the local Life Science Grid (LSG) cluster. The information here applies to *LSG users* who own an account on their *local LSG cluster*:

Contents

- *PBS local jobs*
 - *Introduction*
 - *Quickstart example*
 - * *Preamble*
 - * *Submit a PBS job*
 - *Directives*
 - *System status commands*
 - *Local queues*
 - *How to use local scratch*
 - * *Example with \$TMPDIR*
 - *How to use Grid Storage from the local cluster*

4.2.1 Introduction

The Life Science Grid or LSG is a group of clusters which can be used locally only, or as one big cluster (Grid). Each *local LSG cluster* is part of the Life Science Grid that has its own User Interface (UI) and two Worker Nodes of 64 cores (see [LSG specifications](#)). You can use the local UI for submitting both local *pbs jobs* or *Grid jobs*.

In this section we will focus on the usage of local LSG cluster as a common batch system. The local job submission can be useful when:

- prototyping your Grid application
- running multicore jobs with high number of cores (e.g. more than 8 cores)
- running applications that require just a few jobs to complete. For a large-scale applications that require thousands of analysis to complete, the best option is Grid due to its large compute and storage capacity.

4.2.2 Quickstart example

In this example we will submit a simple PBS (Portable Batch System) job to the local LSG cluster using the fractals example.

Preamble

- Log in to the LSG User Interface, e.g. “ams” cluster (you can find the hostname in the *list of LSG hostnames*):

```
$ssh -X homer@gb-ui-ams.els.sara.nl # replace homer with your username and the UI address of y
```

- Copy the tarball `pbsp_fractals.tar` to your UI directory:

```
$wget http://doc.grid.surfsara.nl/en/latest/_downloads/pbsp_fractals.tar
```

- Copy the fractals source code `fractals.c` to your UI directory.

```
$wget http://doc.grid.surfsara.nl/en/latest/_downloads/fractals.c
```

- Untar the example and check the files:

```
$tar -xvf pbsp_fractals.tar
$cd pbsp_fractals/
$mv ../fractals.c ./
$ls -l

-rw-r--r-- 1 homer homer fractals.c
-rw-rw-r-- 1 homer homer wrapper.sh
```

- Compile the example:

```
$cc fractals.c -o fractals -lm
```

Submit a PBS job

- Submit the job to the local cluster:

```
$qsub wrapper.sh

6401.gb-ce-ams.els.sara.nl
```

This command returns a jobID (6401) that can be used to monitor the progress of the job.

- Monitor the progress of your job:

```
$qstat -f 6401 # replace 6401 with your jobID
```

Optionally, when the job finishes, display the job output image:

```
$convert output "output.png"
$display output.png
```

- List your own jobs:

```
$qstat -u homer # replace homer with your username
```

- Cancel the job you submitted:

```
$qdel 6401 # replace 6401 with your jobID
```

4.2.3 Directives

- Specify the maximum job walltime in hh:mm:ss:

```
##PBS -l walltime=4:00:00 # the job will run 4h at maximum
```

- Specify the number of cores to be allocated for your job:

```
##PBS -l nodes=1:ppn=2 # asks two cores on a single node
```

- The default stdout/stderr target is the directory that you submit the job from. The following line changes the stdout/stderr directory to a specified path (e.g. samples directory):

```
##PBS -e /home/homer/samples/
##PBS -o /home/homer/samples/
```

- Send job status notifications to your email:

```
##PBS -m abe
##PBS -M homer@troy.com #replace with your email
```

4.2.4 System status commands

- List all the running/queued jobs in the cluster:

```
$qstat
```

- Get details for all jobs in a queue, e.g. “long”:

```
$qstat -f long
```

- Show all the running jobs in the system and the occupied cores on the two worker nodes. The very last number in each row (after '/') shows the rank of corresponding core:

```
$qstat -an1
```

- List all running jobs per worker node and core:

```
$pbsnodes
```

4.2.5 Local queues

On the LSG clusters you can find different *queue types*. We recommend you to estimate the walltime of your jobs and specify the queue to send your job. This can be done with the ‘-q’ option in your `qsub` command. For example, if you want to run a job for 72 hours, you need to specify the queue “long”:

```
$qsub -q long wrapper.sh # allow job to run for 72 hours
```

If you don’t specify a particular queue, then your jobs will be scheduled by default on the medium queue (32 hours limit). When the queue walltime is reached, the job will be killed.

See also:

How to run PBS jobs with wallclock greater than 36 hours on the Life Science Grid?

4.2.6 How to use local *scratch*

When you submit a local job, it will land on one of the cluster nodes. This means that the working directory will be different to the directory from where you submit the job (the worker node is a different machine to the UI).

The home UI directory is mounted on the worker node via NFS. For better I/O performance, copy files, computation to the worker node's `/scratch`.

Note: There is an environment variable set on the worker nodes called `$TMPDIR` that points to your job directory, e.g. `/scratch/<jobID>.gb-ui-ams.els.sara.nl/`.

Use `$TMPDIR` in your scripts to locate the `/scratch` directory. The `$TMPDIR` directory also makes sure that any created data is cleaned up properly when the job has finished.

Example with `$TMPDIR`

- Use the `{PBS_O_WORKDIR}` variable to locate your scripts and make sure that your code does not contain any hard-coded paths pointing to your home directory. This variable points to the directory from where you submit the job. Edit the script that you submit with `qsub` as:

```
cd $TMPDIR
cp -r ${PBS_O_WORKDIR}/<your scripts,files> . # note the dot at the end of `cp` command
# ...
# Run the executables
# ...
# When done, copy the output to your home directory:
cp -r $TMPDIR/results ${PBS_O_WORKDIR}/
```

- Submit the script with `qsub`.

4.2.7 How to use Grid Storage from the local cluster

There are many cases that the data that your program needs to run can not be available locally, either because the volume of your home directory is limited or because it is already stored on the *Grid storage*.

Any interaction with the Grid, compute nodes or storage element, requires a *proxy* for your authentication. Even if you run your compute on a local cluster worker node but need to use data from the Grid storage, you will have to *Get a Grid certificate* and *Join a Virtual Organisation*.

To access the Grid storage from jobs submitted locally through `qsub`, you need a valid proxy certificate. However, for local jobs submitted using `qsub` this proxy certificate is not copied automatically.

Therefore, to interact with the Grid storage, you need:

1. A proxy certificate, see *StartGridSession*. You need to do this once, not for each job.
2. To tell the system where the proxy certificate is:
 - Copy your proxy certificate to for example your home-directory using:

```
$cp /tmp/x509up_u39111 /home/homer/ # replace x509up_u39111 with your own proxy file, here "39111"
```

- Set the rights of this file to 600 and treat it as confidential:

```
$chmod 600 /home/homer/x509up_u39111
```


Because your home-directory is shared across the cluster, your proxy will also be available on all nodes within the cluster.

You also need to do this step once every week, and not for each job.

- Tell the system where your proxy certificate is, by setting an environment variable. Add in the job script:

```
$export X509_USER_PROXY=/home/homer/x509up_u39111
```

Now within the job, your *Storage clients* commands will work.

See also:

This section covers the basic usage of PBS jobs particularly on the LSG. For advanced usage of a PBS cluster you may check out the [Lisa batch usage](#) guide or the [NYU Cluster usage](#) guide.

Best practices

5.1 Bootstrap application

When you have a binary program that you want to execute on the Grid you need to create a bootstrap application. This will execute a wrapper script that contains all the necessary information for your job to run. In this page we will show an example to run your bootstrap application on the Grid:

Contents

- *Bootstrap application*
 - *Problem description*
 - *Quickstart example*
 - * *Preamble*
 - * *Run locally*
 - * *Run on the Grid*

5.1.1 Problem description

You want to execute your own binary program on the Grid. When you have a binary program that you want to execute on the Grid you can send it along with the job submission. This can be done when the executable is not too large. The limit is about 100MB, for larger executables you can use *Softdrive* or the *Grid storage*.

Bootstrap basics

See also:

Have a look at our mooc video *Executables on Grid* for a simple example to get started.

To send such an executable along we will use the `InputSandBox` in the job description. The program itself will be executed by a simple shell script (“`wrapper.sh`”). There are several reasons to wrap the call to your executable with a script. One important one is that the executable file might not have executable permissions after it is copied to the Grid worker node. A second is that it is more flexible in the use of input parameters and also to redirect the output. In short, this script provides the correct environment for the execution of the binary.

In this page we will demonstrate a simple bootstrap application using the fractals example.

5.1.2 Quickstart example

Preamble

- Log in to the User Interface (UI):

```
$ssh homer@ui.grid.sara.nl # replace homer with your username
```

- Copy the tarball `bootstrap_fractals.tar` to your UI directory.
- Copy the fractals source code `fractals.c` to your UI directory.
- Untar the example and check the files:

```
$tar -xvf bootstrap_fractals.tar
$cd bootstrap_fractals/
$mv ../fractals.c ./
$ls -l

-rw-r--r-- 1 homer homer fractals.c
-rw-rw-r-- 1 homer homer fractals.jdl
-rw-rw-r-- 1 homer homer wrapper.sh
```

- Compile the example:

```
$cc fractals.c -o fractals -lm
```

Warning: It is advisable to compile your programs on the User Interface (UI) Machine. The Grid nodes have similar environments and the chance of your job to run successfully on a remote worker node is larger when your program is able to run on the UI.

Run locally

- Run the example locally on the UI with a set of parameters to understand the program:

```
$. /fractals -o output -q 0.184 -d 2280 -m 4400 # try different parameters, e.g. -q 0.184 -d 2280
```

This will take a while, depending on the input parameters you selected. Once finished, it will create the “output” file.

- Convert the output file to .png format and display the picture:

```
$convert output "output.png"
$display output.png
```

Run on the Grid

- Create a proxy valid for a week:

```
$startGridSession lsgrid # replace lsgrid with your VO
```

- Inspect the JDL file `fractals.jdl`:

```
Type = "Job";
JobType = "Normal";
Executable = "/bin/sh";
Arguments = "wrapper.sh";
StdOutput = "stdout";
```

```
StdError = "stderr";
InputSandbox = {"wrapper.sh", "fractals"};
OutputSandbox = {"stdout", "stderr", "output"};
```

In the JDL file we specify the content of the in- and output sandboxes. These sandboxes allow you to transfer small files to or from the Grid. The input sandbox contains all the files that you want to send with your job to the worker node, like e.g. the fractals script that you want executed. The output sandbox contains all the files that you want to have transferred back to the UI, e.g. the output fractals image.

- Inspect the contents of the `wrapper.sh` script:

```
$ cat wrapper.sh
#!/bin/bash
chmod u+x fractals
./fractals -o output -q 0.184 -d 2280 -m 4400
...
```

Once this jobs lands on the Grid, it will execute the `wrapper.sh` script which is a master script to set the program environment and initiate the program execution. In the `wrapper.sh` script you may include also the commands to retrieve input from a Grid storage location or transfer the output results to a Grid storage location.

- Submit the job to the Grid:

```
$glite-wms-job-submit -d $USER -o jobIds fractals.jdl
```

- Check the job status from command line on the UI:

```
glite-wms-job-status https://wms2.grid.sara.nl:9000/6swP5FEfGVZ69tVB3PwnDQ #replace with your job ID
# or
glite-wms-job-status -i jobIds
```

- Once the job is finished, get the job output to the UI:

```
$glite-wms-job-output --dir . -i jobIds
```

- Convert the output file to .png format and display the picture:

```
$convert homer_6swP5FEfGVZ69tVB3PwnDQ/output "output.png" # replace with your job output directory
$display output.png
```

5.2 Parametric jobs

The *pilot jobs* use the technique of parametric jobs for the job submission to the Grid. In this page we give an example of parametric jobs:

Contents

- *Parametric jobs*
 - *About*
 - *Example*

5.2.1 About

Pilot jobs are submitted to the Grid with a specific *Job Description Language* type called `Parametric`. A parametric job causes a set of jobs to be generated from one JDL file.

5.2.2 Example

In the example below, the parametric job will create 3 child jobs (see line 4) that will all run the same executable (see line 6). The value `_PARAM_` will be replaced by the actual value of Parameters during the JDL expansion.

`ParameterStart` defines the starting value for the variation, `ParameterStep` the step for each variation and `Parameters` defines the value where the submission of jobs will stop (that value itself is not used) . The number of jobs is: $(Parameters - ParameterStart) / ParameterStep$

- Log in to your User Interface.
- Create a file with the following content describing the job requirements. Save it as `parametric.jdl`:

```
1  JobType = "Parametric";
2  ParameterStart=0;
3  ParameterStep=1;
4  Parameters=3;
5
6  Executable = "/bin/hostname";
7  Arguments = "-f";
8  StdOutput = "std_PARAM_.out";
9  StdError = "std_PARAM_.err";
10 OutputSandbox = {"std_PARAM_.out", "std_PARAM_.err"};
```

- You can submit the parametric job as any Grid job:

```
$glite-wms-job-submit -d $USER -o jobIds parametric.jdl
```

In this case, 3 child jobs will be generated. Each job will generate two files: `std0.out` and `std0.err`, `std1.out` and `std1.err`, `std2.out` and `std2.err`.

- Monitor the job status to see the the parent job URL and the 3 child jobs URLs with their status:

```
$glite-wms-job-status -i jobIds

===== glite-wms-job-status Success =====
BOOKKEEPING INFORMATION:

Status info for the Job : https://wms2.grid.sara.nl:9000/3ii77PlaSSTKue-MkT_y9g
Current Status:      Running
Submitted:           Sat Jan 4 12:54:56 2016 CET
=====

- Nodes information for:
  Status info for the Job : https://wms2.grid.sara.nl:9000/0OZYR142AXspdm807L6YWA
  Current Status:      Running
  Status Reason:       unavailable
  Destination:         ce.lsg.bcbr.uu.nl:8443/cream-pbs-express
  Submitted:           Sat Jan 4 12:54:56 2016 CET
=====

  Status info for the Job : https://wms2.grid.sara.nl:9000/9uO8Hp6H3qCBAK3abx7G4A
  Current Status:      Running
```

```
Status Reason:      unavailable
Destination:       gb-ce-amc.amc.nl:8443/cream-pbs-express
Submitted:         Sat Jan 4 12:54:56 2016 CET
```

```
=====
```

```
Status info for the Job : https://wms2.grid.sara.nl:9000/CVYq7F6lqokBvJvsfU4ELw
Current Status:    Running
Status Reason:     unavailable
Destination:       gb-ce-lumc.lumc.nl:8443/cream-pbs-express
Submitted:         Sat Jan 4 12:54:56 2016 CET
```

```
=====
```

This is just an example. In practice you shouldn't send more than **50** jobs this way (Parameters=50). The parametric jobs is the technology used for submitting the pilot jobs. There is no need to monitor their status or retrieve the job output through the WMS as the *pilot frameworks* will take care of this.

5.3 ToPoS Overview

This page is about the ToPoS pilot framework:

Contents

- *ToPoS Overview*
 - *About ToPoS*
 - * *Security*
 - * *File size limitations for Tokens*
 - *An example*
 - *Source Code*
 - *ToPoS clients*

5.3.1 About ToPoS

ToPoS is a system that implements *pilot jobs*. It is a very simple system, which can be very effective. The name of ToPoS refers to Token Pools. The idea is that all a task server needs to provide to pilot jobs is a token which uniquely identifies a task. This token can be as simple as a unique number. All the Pilot job has to do is to map the token to a task, execute it and report back to the token pool server. Then Grid computing comes down to creating lists of tasks, and present them as tokens in a pool.

The user has to create tokens and to submit pilot jobs. The pilot jobs will contact the Token Pool Server and request for a token. The Token Pool Server will hand out a unique token to each requesting job. Of course a pilot job will have to know what to do with the token. The simplest form a token can take is a number. In that case, the pilot job will have to map the number to a specific task it has to compute. When it finishes this task it will delete the token from the Token Pool on the server.

The idea of tokens in a pool can be extended in several ways. Tokens can be files, parameters, numbers etc. What is important about the concept of a token, is that it somehow identifies a unique task to be computed. For example, consider a researcher who wants to align 100 genome sequences against a database using a certain program P. ToPoS can be used as follows. The 100 genome sequences are not very large in size and can be uploaded as tokens in ToPoS. The user can do this with an Internet browser. The database can be expected to be large and it is usually recommended to upload this to a Storage Element and then replicate it several times. He then submits about a hundred or more pilot jobs, each containing the program P and a reference to the database. The job will also contain a command to request a token from ToPoS. ToPoS will get requests from running jobs for tokens (in this case genome sequences). It deals

these out uniquely. When a job finishes its computation it will tell ToPoS to delete the token and it will then request a new one. ToPoS will deal out tokens as long as there are tokens in the pool. In this scenario, it is possible for jobs to receive the same token and that a computation is performed more than once. In general, this is not a problem. However, it is possible to “lock” tokens. When tokens are locked, they will only be dealt out once. Each lock comes with a timeout which means that before the expiration of this timeout a job should confirm possession of the token. If the timeout expires the lock will be removed and the token will be free for distribution.

You can have a peek at ToPoS here: <http://topos.grid.sara.nl/4.1/> or here: <http://purl.org/sara/topos/latest/>

Security

A short note about ToPoS and security:

Please be aware that the `topos.grid.sara.nl` is a public service that on purpose employs a security through obscurity model. Private and sensitive data shouldn't be put in your pools.

Anyone knowing the random urls can read and modify your tokens.

File size limitations for Tokens

Although it is possible to upload small files as tokens, be aware that ToPoS is made to process 100's of request per second; big tokens can be a bottleneck for this performance. Many large files should be put at Grid storage or at a storage location of your choice; putting only filenames in ToPoS can be a helpful approach.

Critical token size is between 100KB and 10MB.

5.3.2 An example

We have an example available on this page *[Topos Example](#)*

5.3.3 Source Code

The source code of ToPoS is on [GitHub ToPos](#).

5.3.4 ToPoS clients

ToPoS Bash client

This page contains information about `BashTopos`, a bash client for ToPoS:

Contents

- *ToPoS Bash client*
 - *Introduction*
 - * *Function description*

Introduction

There is a bash ‘library’ available to ease the communication with ToPoS. It is written in bash script and requires curl and awk to be present. It has been tested on CentOS 6.7 and Ubuntu 14.04. You can find it [here](#).

To use it, you need to make the file executable:

```
$chmod +x topos
```

After that you can call the function:

```
USAGE: ./topos [ command ] [ arguments ]
```

Where the combination command and arguments is one of:

```
newPool
createTokensFromLinesInFile [POOLNAME] [FILENAME]
uploadFileAsToken [POOLNAME] [FILENAME]
uploadFilesInDirAsTokens [POOLNAME] [DIRNAME]
nextToken [POOLNAME]
nextTokenWithLock [POOLNAME] [TIMEOUT]
getToken [POOLNAME] [TOKENNAME]
refreshLock [POOLNAME] [LOCKNAME] [TIMEOUT]
deleteLock [POOLNAME] [LOCKNAME]
deleteToken [POOLNAME] [TOKENNAME]
```

Function description

newPool

Returns a new, unused pool name.

Usage: newPool

Returns the unique new pool name

createTokensFromLinesInFile

Creates a token for each line in a text file.

Usage: createTokensFromLinesInFile [POOLNAME] [FILENAME]

Returns nothing

uploadFileAsToken

Creates a token with the contents of a file.

Usage: uploadFileAsToken [POOLNAME] [FILENAME]

Returns the token name

uploadFilesInDirAsTokens

Creates tokens for each file in a directory.

Usage: uploadFilesInDirAsTokens [POOLNAME] [DIRNAME]

Returns nothing

nextToken

Fetches the next token.

Usage: nextToken [POOLNAME]

Returns the token name (NOT its contents!)

nextTokenWithLock

Fetches the next token and puts a lock on it. The lock description is the hostname.

Usage: `nextTokenWithLock [POOLNAME] [TIMEOUT]`

Returns the token name (NOT its content!) and on the next line the lock name

getToken

Fetches the content of a token.

Usage: `getToken [POOLNAME] [TOKENNAME]`

Returns the content of the token

refreshLock

Refreshes a lock so it doesn't time out.

Usage: `refreshLock [POOLNAME] [LOCKNAME] [TIMEOUT]`

Returns nothing

deleteLock

Deletes a lock so the associated token becomes available again.

Usage: `deleteLock [POOLNAME] [LOCKNAME]`

Returns nothing

deleteToken

Deletes a token.

Usage: `deleteToken [POOLNAME] [TOKENNAME]`

Returns nothing

Perl ToPoS

This page contains information about `PerlToPoS`, a perl client for ToPoS:

Contents

- *Perl ToPoS*
 - *Introduction*
 - *Obtaining PerlToPoS*
 - *Writing ToPoS Clients*
 - * *General script structure for shared tokens*
 - * *General script structure for exclusive tokens*
 - * *Opening an existing pool*
 - * *Getting the next token*
 - * *Getting the token contents*
 - *Getting the content of a plain text token object*
 - *Getting the file contained in a file token object*
 - * *Renewing token locks*
 - * *Deleting tokens*
 - *Manipulating ToPoS pool*
 - * *Creating a new pool with a random name*
 - * *Saving the pool name to a file*
 - * *Populating a pool with tokens*
 - *Creating text tokens individually*
 - *Creating multiple tokens from a file*
 - *Creating a file token*
 - * *Deleting a pool*
 - * *Getting the pool name*
 - *Complete example*
 - * *Creating a script to populate a new pool with tokens*
 - * *Creating a pilot job*
 - * *Creating a job submission file*

Introduction

PerlToPoS is a ToPoS client library written in Perl. The library has no dependencies other than those found on all Grid nodes, so including the two modules ‘ToposPool.pm’ and ‘ToposToken.pm’ is enough to run pilot jobs using PerlToPoS.

As an introduction, the following example shows a simple, but functional PerlToPoS client:

```
use ToposPool;
use ToposToken;

# open an existing pool, assumed to be filled with tokens
my $pool = ToposPool -> new ("my_pool");

while (my $token = $pool -> next_token) {

    # process the token, which can contain text or a file,
    # store the results
    ...

# when processed, remove the token
    $token -> delete;
}

# no more tokens
```

While the example seems simple and uses no advanced feature like locking tokens, it shows most of what is needed to use ToPoS in Perl scripts. Moreover, it shows that all HTTP commands have been hidden in the objects and methods.

Obtaining PerlToPoS

The PerlToPoS client libraries can be downloaded from [GitHub PerlTopos](#).

Writing ToPoS Clients

General script structure for shared tokens For scripts which use shared tokens, no locking mechanism is used, making the script easy. In this case, the structure of pilot jobs will be:

1. include the library modules
2. **open an existing pool, either from:**
 - a known pool name, or
 - a pool name stored in a file
3. **while there are more tokens:**
 - (a) get the next token
 - (b) process the token contents
 - (c) store the results
 - (d) **if the token was successfully processed and the results stored:**
 - delete the token

General script structure for exclusive tokens If tokens are to be exclusive, tokens must be locked and optionally renewed while the computation is still running. A possible structure is then:

1. include the library modules
2. **open an existing pool, either from:**
 - a known pool name, or
 - a pool name stored in a file
3. **while there are more tokens:**
 - (a) get the next token, locking it
 - (b) while not finished:
 - i. process the token contents
 - ii. renew the lock
 - (c) store the results
 - (d) **if the token was successfully processed and the results stored:**
 - delete the token

Opening an existing pool To open an existing pool with a known name (or simple name), use the ‘new’ method:

```
my $pool = ToposPool -> new('name');
```

This method returns a pool object which has various methods for manipulating the ToPoS pool.

If the pool name was saved in a file (see saving the pool name to a file), the ‘load’ function can be used to read the pool name from a file and open the existing pool with that name. This is again convenient if after populating a new pool with tokens, the pool name was saved to a file. The command is:

```
my $pool = ToposPool -> load('my_pool_file.txt');
```

If no pool file is specified, the file is assumed to be ‘pool_id.txt’, which is also the default for saving pools, see populating pools.

Getting the next token After opening an existing pool, tokens objects can be retrieved from that pool with the ‘next_token’ method:

```
my $token_object = $pool -> next_token; # no lock
```

If no arguments are specified, the token is not locked. If an optional argument is specified, it the token is locked for the specified duration in seconds:

```
my $locked_token = $pool -> next_token(60); # lock for 60 seconds
```

The ‘next_token’ method returns a ToposToken object (not text!), which can be further inspected using the methods below.

If there are no more tokens, or if all remaining tokens are locked, ‘next_token’ returns ‘undef’, so it can be used in a ‘while’ construct as shown in the introduction.

Getting the token contents Tokens can contain plain text or a file, depending on what was stored in the token when it was created. To find out what the token contains, use the ‘is_file’ method:

```
if ($token_object -> is_file) {
    # token is a file
    ...
}
else {
    # token is plain text
    ...
}
```

Getting the content of a plain text token object If a token object contains plain text, the text can be retrieved using the ‘content’ method:

```
my $token_content = $token_object -> content;
```

Getting the file contained in a file token object If a token object contains a file, there are two convenient methods:

- ‘filename’ which returns the name of the file when it was uploaded, but without any path information;
- ‘save’, which will save the file in the current directory (as a safety feature), with the original file name or with the specified file name.

Both methods can be used as follows:

```
if ($token_object -> is_file) {  
    $token_object -> save;  
    process_file ($token_object -> filename);  
}
```

where ‘process_file’ is assumed to be some routine responsible for the actual processing, taking a file name as an argument.

The ‘save’ method has an optional argument which stores the file under the given name:

```
# does not use the original name  
$token_object -> save('my_file.dat');
```

Renewing token locks Locks on tokens can be renewed using the ‘renew_lock’ method, which has an optional timeout. If no timeout is specified, the timeout of the previous lock is reused:

```
$token_object -> renew_lock;           # same timeout as previous lock  
$token_object -> renew_lock(600);      # 600 second / 10 minute lock renewal
```

Deleting tokens After successful processing and storing the results, the token must be deleted from the token pool - otherwise tokens will be recycled and your pilot job will never finish!

Deleting a token is done using the ‘delete’ method on a token:

```
$token_object -> delete;
```

Manipulating ToPoS pool

In client scripts, pool objects are only used to get next tokens. In preparation scripts, the methods of a pool object can be used in scripts to manipulate the pool itself, for example to populate the pool with tokens.

Creating a new pool with a random name A new pool with a random name is created using the ‘new’ method without any arguments:

```
my $pool = ToposPool -> new();
```

Saving the pool name to a file To avoid having to copy-and-paste a random pool name, the pool name can be saved to a file using the ‘save’ method. The method takes one optional argument, the file name. If no file name is specified, the pool name is saved in a file called ‘pool_id.txt’:

```
$pool -> save("my_pool_id.txt");  
$pool -> save; # saves the pool name in 'pool_id.txt'
```

The pool can be opened again using the ‘load’ method, again with an optional file name:

```
my $pool = ToposPool -> load("my_pool_id.txt");  
my $pool = ToposPool -> load; # loads the pool from 'pool_id.txt'
```

The file containing the pool name can be used in the InputSandbox in Grid jobs, making it easy to reopen the ToPoS pool from a pilot job.

Populating a pool with tokens There are three (currently supported) ways of populating a pool with tokens:

1. creating text tokens individually
2. creating multiple text tokens from a file
3. creating file tokens

Creating text tokens individually To create a token containing text, use the ‘create_token’ method:

```
# create text one token
$pool -> create_token('abc def');
```

Creating multiple tokens from a file Multiple tokens can be created from a file if each line in the file contains exactly one token, using the ‘create_tokens_from_file’ method:

```
# create multiple tokens; assume that the file 'input_file_names.txt'
# contains a list of input file names which must be processed
$pool -> create_tokens_from_file('input_file_names.txt');
```

Creating a file token To create a file token, use the ‘upload_file_as_token’ method:

```
# upload 'input_1.dat', which will become a token
$pool -> upload_file_as_token('input_1.dat');
```

Deleting a pool A token pool can be deleted, effectively removing all tokens in the pool, using the ‘delete’ method on a pool:

```
# remove all tokens in the pool
$pool -> delete;
```

Getting the pool name If you need to know the name of the pool, use the ‘name’ method:

```
my $pool_name = $pool -> name;
```

Complete example

As a complete example, the following scripts will first populate a new pool with numbers; the pilot job will then compute the square of the numbers as a processing stage and store the results in a second pool.

Creating a script to populate a new pool with tokens The first script populates a new pool with tokens, each of which contains a number. With the two PerlToPoS modules ‘ToposPool.pm’ and ‘ToposToken’ in a fresh directory, create the script:

```
#!/usr/bin/perl

use ToposPool;
use ToposToken;
use strict;
use warnings;

my $pool = ToposPool -> new ("example_input_pool");
```

```
# fill the pool with numbers from 1 through 100
for my $i (1..100) {
    $pool -> create_token ($i);
}

# done
```

Run the script - it should run in a few seconds.

After running the script you can verify that the pool was indeed filled with tokens by browsing

http://topos.grid.sara.nl/4.1/pools/example_input_pool

Creating a pilot job We used a pool named “input_pool” for storing the data which must be processed. For simplicity, let’s call the pool with results “output_pool”. The script for the pilot job is:

```
#!/usr/bin/perl

use ToposPool;
use ToposToken;
use strict;
use warnings;

my $input_pool = ToposPool -> new ("example_input_pool");
my $output_pool = ToposPool -> new ("example_output_pool");

# process input tokens until there are no more tokens

# lock the token for 3 seconds
while ( my $token = $input_pool -> next_token(3) ) {

    # get the text contained in the token
    my $n = $token -> content;

    # 'process' the input data
    my $n_squared = $n * $n;

    # store the results
    my $result = sprintf("The square of %d is %d", $n, $n_squared);

    $output_pool -> create_token ($result);

    # delete the token
    $token -> delete;
}

# done
```

Save the script as ‘example_pilotjob.pl’.

Note that the timeout for a task is set to 3 seconds. It is expected that each task, so processing and storing data, should take no longer than 1 second; the largest delay is in the network traffic to and from ToPoS, and even 1 second is pessimistic. If however some task fails in the processing (in this case very unlikely, but not unlikely in real-world cases) or in the storing phase (real possibility, due to network hiccups), the token is unlocked and available to other pilot jobs.

Creating a job submission file The job submission file is a regular *Job Description Language* file with the following properties:

- the job type must be `Parametric`
- the number of parameters is the number of machines that should be used per job submit
- the executable must be `/usr/bin/perl`
- the first argument must be the name of the script; so in the example above, the argument is `example_pilotjob.pl`
- the input sandbox must at least contain the two PerlToPoS perl modules and the name of the pilot job script

We create a job submission file which will start the processing on 5 nodes. Replace <your VO name> with the name of your Virtual Organisation.

```
# example JDL file for the square computation
Type = "Job";
JobType = "Parametric";
VirtualOrganisation = "<your VO name>";
DefaultNodeShallowRetryCount = 5;

# Parametrization.
Parameters = 5;
ParameterStart = 1;
ParameterStep = 1;

# Specify the input data ...
InputSandbox = {"ToposToken.pm",
                "ToposPool.pm",
                "example_pilotjob.pl" };

# Executable and arguments
Executable = "/usr/bin/perl";
Arguments = "example_pilotjob.pl";

# Specify output data ...
StdOutput = "stdout_PARAM_.log";
StdError = "stderr_PARAM_.log";

OutputSandbox = {"stdout_PARAM_.log",
                 "stderr_PARAM_.log" };
```

You can submit the job to start the processing. The results are stored back in ToPoS, in

http://topos.grid.sara.nl/4.1/pools/example_output_pool/tokens

Each token contains a result, which you can verify by browsing the tokens.

Note that each pilot job will process the available work. This means that you can submit the job multiple times, each time requesting 5 cores. If more cores are available, this speeds up the processing. When all work is done, the jobs simply quit. This is useful for tasks with longer processing.

- Bash wrapper: *ToPoS Bash client*
- Perl client: *PerlToPoS*

Besides these, 'wget' and 'curl' can be used in shell scripts or even other languages, to send 'raw' HTTP-commands. However, this can get tedious and is error-prone. Client libraries have been written which make accessing ToPoS easier. If possible, we advise you to use one of the above client libraries.

5.4 Topos Example

This page presents a ToPoS pilot job example:

Contents

- *Topos Example*
 - *Overview of the example*
 - *Quick overview of ToPoS*
 - *ToPoS sample client*
 - *Running the example*
 - * *Creating a parameter file for the fractals program*
 - * *Getting a unique ToPoS poolname*
 - * *Creating ToPoS tokens*
 - * *Running the example*
 - * *Retrieve the output*

5.4.1 Overview of the example

An application called “fractals” needs to be executed in parallel a certain amount of times. Each time, the program is called with a different set of parameters. The parameters for all of these tasks are saved in a file in which a single line contains parameters for a single task.

Using the combination Grid, pilot jobs and the ToPoS service allows the user to run and finish all tasks without having to bother with failures and re-submissions, and make more efficient use of the Grid while doing so.

The pipeline of the job is as follows. We upload the file that contains our parameters to the ToPoS service. After that we submit a parametric Grid job that, in an endless loop:

- asks ToPoS for the next line
- tells ToPoS not to give that line to anybody else as long as it works on it
- calls the fractals program with that line of parameters
- if successful, tells ToPoS to delete that line
- otherwise, tells ToPoS it can give out that same line again
- continues at the beginning of the loop

5.4.2 Quick overview of ToPoS

ToPoS, however, does not work with the notion of lines but rather with a notion of a set (called a “pool”) and items within that set (called “tokens”). ToPoS is a RESTful webservice, and in order to use it, it is helpful to have a notion of the REST-style of software architecture. You can find the reference document for the 4.1 version of ToPoS at [ToPoS Reference Manual](#).

In short, you upload (a set of) information to the ToPoS service, which it makes available for download under a unique token URL. Each token URI exists only in the namespace of a certain pool URI - in other words, each token is part of a pool. The system allows you to ask for the next available token in a certain pool, and optionally lock that token for a certain time. If a token is locked, it means it will not be given out by the service for the specified amount of time. A lock in itself can also be managed through a unique lock URL, meaning you can delete or prolong the lock, if that should be needed.

5.4.3 ToPoS sample client

This example requires a ToPoS library implementing a subset of ToPoS' features. It is written in Bash script and requires `curl` and `awk` to be present. It has been tested on CentOS 6.7 and Ubuntu 14.04. You can find the documentation (including download location) on this library at the [ToPoS library for Bash](#) page.

The ToPoS service offers the possibility to upload a text file of which each line will be made a token. We will use this functionality to make a single token of each line in our file with parameters. This way, each token represents a task that needs to be executed on the Grid.

5.4.4 Running the example

Start by downloading and unpacking the necessary files.

- Log in to the User Interface (UI):

```
ssh homer@ui.grid.sara.nl
# replace homer with your username
```

- Copy the tarball `pilot_topos_fractals.tar` to your UI directory.
- Copy the fractals source code `fractals.c` to your UI directory.
- Copy the topos bash client `topos` to your UI directory.
- Untar the example and check the files:

```
$star -xvf pilot_topos_fractals.tar
$cd pilot_topos_fractals/
$mv ../fractals.c ./
$mv ../topos ./
$chmod +x topos
$ls -l

-rwxr-xr-x 1 homer homer  convert
-rwxr-xr-x 1 homer homer  createFractalsFromTokens
-rwxr-xr-x 1 homer homer  createTokens
-rw-rw-r-- 1 homer homer  fractals.c
-rw-r--r-- 1 homer homer  fractals.jdl
-rw-r--r-- 1 homer homer  README
-rwxrwxr-x 1 homer homer  topos
```

- Compile the example:

```
$cc fractals.c -o fractals -lm
```

Warning: It is advisable to compile your programs on the User Interface (UI) Machine. The Grid nodes have similar environments and the chance of your job to run successfully on a remote worker node is larger when your program is able to run on the UI.

Creating a parameter file for the fractals program

This example includes a bash script (`./createTokens`) that generates a sensible parameter file, with each line representing a set of parameters that the fractals program can be called with. Without arguments it creates a fairly sensible set of 24 lines of parameters. You can generate different sets of parameters by calling the program with a combination of `-q`, `-d` and `-m` arguments, but at the moment no documentation exists on these. We recommend not to use them for the moment.

After you ran the `createTokens` script you'll see output similar to the following:

```
$ ./createTokens
/tmp/tmp.fZ33Kd8wXK
```

Getting a unique ToPoS poolname

In order to run the tasks we first need to have the ToPoS service create tokens for us, based on the lines in our generated parameter file. Since all tokens need to be part of a pool, we first need to find out a suitable poolname. You can choose anything you like here, but the only way to be sure the poolname does not yet exist within ToPoS and to avoid clashes, we can ask the service for a unique poolname by calling:

```
$ ./topos newPool
f24c058fdb6793ed7b6d5ff9
```

Note that the poolname does not end with a newline in order to make it easier usable by scripts.

Creating ToPoS tokens

Now that we have a poolname, either thought of by ourselves or by the ToPoS service, we can upload the file to the service and have it create tokens:

```
$ ./topos createTokensFromLinesInFile f24c058fdb6793ed7b6d5ff9 /tmp/tmp.fZ33Kd8wXK
```

You might see some HTML output that you can ignore. To check if the request went well you can have a look at your pool by querying the service from a browser. Point it at [https://topos.grid.sara.nl/4.1/pools/\[POOLNAME\]/](https://topos.grid.sara.nl/4.1/pools/[POOLNAME]/) and check that it contains tokens by looking under the Content section.

Running the example

Now that the tokens are uploaded we can submit a Grid job. A sample JDL file, submitting 10 jobs at once, is included. You still need to fill in the poolname you use in this file by replacing the placeholder [POOLNAME]. It will call the `./createFractalsFromTokens` script, which is the implementation of a simple pilot job that implements the pipeline as described above.

This script calls the fractals program. You can compile it by simply running:

```
$ cc fractals.c -o fractals -lm
```

To have an impression of how `./createFractalsFromTokens` works you can call it on a local Linux machine (providing it can run the topos client and the fractals program):

```
$ ./createFractalsFromTokens -p [POOLNAME]
```

It will recursively generate an image based on parameters received from the specified ToPoS pool, and output the path to the generated image.

You can also submit the JDL file (don't forget to edit it to include your poolname!) to the Grid and have all tokens processed in parallel. You will be able to see the progress by querying ToPoS through your browser and checking the amount of locks that exist, as well as the amount of tokens that are left.

Retrieve the output

To check if the output is ready you can have a look at your pool by querying the service from a browser. Point it at [https://topos.grid.sara.nl/4.1/pools/\[POOLNAME\].output/](https://topos.grid.sara.nl/4.1/pools/[POOLNAME].output/) and check that it contains the output of the tokens by looking under the Content section.

Note that for this example, we made the pilot job upload the results to another token pool with the same name as the original token pool and ‘.output’ appended to it. However, this is not default ToPoS functionality, but done for the sake of keeping the example as simple as possible. In a normal situation, you’ll almost always want to transfer the generated image (or whatever output you have) to a storage element or external storage using a supported protocol.

5.5 Picas Overview

This page is about the PiCaS pilot framework:

Contents

- *Picas Overview*
 - *About PiCaS*
 - * *Source Code*
 - *Picas server*
 - *Picas client*
 - * *How to use*
 - *Grant access on Picas*
 - *Background: CouchDB*
 - * *Picas views*

5.5.1 About PiCaS

Let’s say that you have a number of *tasks* to be processed on the Grid Worker Nodes. Each task requires a set of parameters to run. The parameters of each task (run id, input files, etc) construct an individual piece of work, called *token*, which is just a description - not the task itself.

The central repository for the tasks is PiCaS, a Token Pool Server. The pilot jobs request the next free token for processing from PiCaS. As said, the content of the tokens is opaque to PiCaS and can be anything your application needs.

PiCaS works as a queue, providing a mechanism to step through the work one token at a time. It is also a pilot job system, indicating that the client communicates with the PiCaS server to fetch work, instead of having that work specified in a jdl (or similar) file.

The server is based on [CouchDB](#) (see [CouchDB book](#)), a NoSQL database, and the client side is written in Python.

Source Code

The source code of Topos is on [Github Picas Client](#).

5.5.2 Picas server

More about Picas Server?

See also:

Check out our mooc videos Picas server side [Part I](#) and [Part II](#).

On the server side we have a queue which keeps track of which tokens are available, locked or done. This allows clients to easily retrieve new pieces of work and allows also easy monitoring of the resources. As every application needs different parameters, the framework has a flexible data structure that allows users to save different types of data. Because of this, tokens can contain any kind of information about a task: (a description of) the work that needs to be done, output, logs, a progress indicator etc.

The server is a regular CouchDB server with some specific views installed. For more information on this, see the [Background: CouchDB](#) section.

5.5.3 Picas client

More about Picas Client?

See also:

Check out our mooc video [Picas client side](#)

The PiCaS client library was created to ease communication with the CouchDB back-end. It allows users to easily upload, fetch and modify tokens. The system has been implemented as a Python Iterator, which means that the application is one large for loop that keeps on running as long as there is work to be done. The client is written in Python and uses the `python couchdb` module, which requires at least python version 2.6. The client library is constructed using a number of modules and classes, most important of these are:

- The *Actors* module contains the `RunActor` class. This is the class that has to be overwritten to include the code that calls the different applications (tasks) to run.
- The *Clients* module contains a `CouchClient` class that creates a connection to CouchDB.
- The *Iterators* module contains classes responsible for working through the available tokens. The `BasicViewIterator` class assumes that all the token information is encoded directly into the CouchDB documents.
- The *Modifiers* module is responsible for modification of the PiCaS tokens. It makes sure the tokens are modified in a uniform way.

How to use

The source code includes a simple example of how to use PiCaS. You'll have to write your pilot job in Python. Using PiCaS boils down to these steps:

1. Extend the `RunActor` class and overwrite the necessary methods:

- `process_token`; should be implemented and contain the code to process your task.
- `prepare_env`; is called only once, before the first run of this pilot job. For example, this is a good place to download a database that will be used by all tasks.
- `prepare_run`; is called before each task. For example, this is a good place to create output directories or open database connections

- `cleanup_run`; is called after each task. For example, this is a good place to clean up anything setup in `prepare_run`. Note that it is not guaranteed that this code will be run. For example in the case of a failure during `process_token`
 - `cleanup_env`; is called only once, just before the pilot job is shutting down. For example, this is a good place to clean up anything setup during `prepare_env` (remember to clean up your temporary files when you leave!). Note that it is not guaranteed that this code will run as the job can be interrupted or crash before this.
2. Instantiate a `CouchClient`
 3. Instantiate a `BasicTokenModifier`
 4. Instantiate a `BasicViewIterator` (and provide it the `CouchClient` and `BasicTokenModifier`)
 5. Instantiate your `RunActor` subclass (and provide it the `BasicViewIterator` and `BasicTokenModifier`)
 6. Call the `run()` method of your `RunActor` to start processing tokens

For another example, see the *Picas Example* in our documentation.

5.5.4 Grant access on Picas

Any user with a Grid project and allocated *quotas* can get a PiCaS account and also obtain a database on the CouchDB server. If you want access, just contact us at helpdesk@surfsara.nl to discuss your design implementation and request your PiCaS credentials.

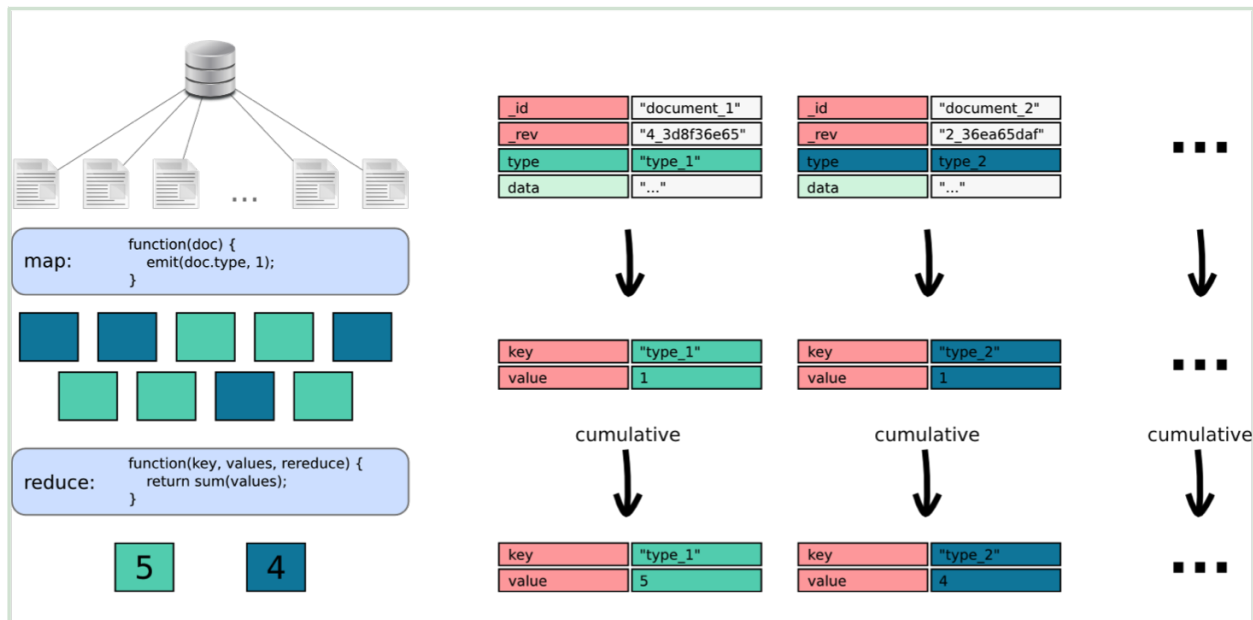
5.5.5 Background: CouchDB

PiCaS server is based on CouchDB. CouchDB stores documents which are self-contained pieces of information. These documents support a dynamic data model, so unlike traditional databases, CouchDB allows storing and retrieving any piece of information as long as it can be defined as key-value pairs. This feature is used to store all the information needed to keep track of the job stages and all of the required in- and outputs.

CouchDB also provides a Restful HTTP API, which means that we can easily access information with an HTTP client. This can be a browser, a command-line application like `curl` or a complete client library. It is also possible to interact with the CouchDB database behind PiCaS using the web-interface.

Picas views

CouchDB views are the basic query mechanism in CouchDB and allow you to extract, transform and combine data from different documents stored in the same database. This process is based on the Map/Reduce paradigm. In the case of CouchDB, the Map step takes every document from a database and applies a piece of code. It then sorts the output of that step based on the key that you supply and give it to the reducer. The code you supply for the reducer combines data from the mapper that have the same key.



The map code works on a 'per document' basis, so every document is run through that code one by one. The `emit` statement returns the value to the `reduce` command, again, this is all done for every document. In this case we are only interested in the type of the document, and as we want to count how many of each type there are, we provide the type as the key for the `emit` statement.

5.6 Picas Example

This page presents a PiCaS pilot job example:

Contents

- *Picas Example*
 - *Problem description*
 - * *Prerequisites*
 - *Picas sample example*
 - * *Prepare your Tokens*
 - *Create the Tokens*
 - *Upload your Tokens to the PiCaS server*
 - *Run the example locally*
 - *Run the example on the Grid*
 - *Checking failed jobs*

5.6.1 Problem description

More about Picas in practice?

See also:

Check out our mooc videos Picas examples *Part I* and *Part II*.

In this example we will implement the following pilot job workflow:

- First we define and generate the application tokens with all the necessary parameters.
- Then we define and create a shell script to process one task (*process_task.sh*) that will be sent with the job using the input sandbox. This contains some boiler plate code to e.g. setup the environment, download software or data from the Grid storage, run the application etc. This doesn't have to be a shell script, however, setting up environment variables is easiest when using a shell script, and this way setup scripts are separated from the application code.
- We also define and create a Python script to handle all the communication with the token pool server, call the *process_task.sh* script, catch errors and do the reporting.
- Finally we define the JDL on the User Interface machine to specify some general properties of our jobs. This is required to submit a batch of pilot jobs to the Grid that will in turn initiate the Python script as defined in the previous step.

Prerequisites

To be able to run the example you must have:

- All the three Grid *Prerequisites* (User Interface machine, Grid certificate, VO membership)
- An account on PiCaS server (send your request to <helpdesk@surfsara.nl>)

5.6.2 Picas sample example

- Log in to the UI and download the *pilot_picas_fractals.tgz* example, the couchdb package for Python *couchdb.tgz* and the fractals source code *fractals.c*.
- Untar *pilot_picas_fractals.tgz* and inspect the content:

```
$tar -zxf pilot_picas_fractals.tgz
$cd pilot_picas_fractals/
$ls -l
-rwxrwxr-x 1 homer homer 1247 Jan 28 15:40 createTokens
-rw-rw-r-- 1 homer homer 1202 Jan 28 15:40 createTokens.py
-rw-rw-r-- 1 homer homer 2827 Jan 28 15:40 createViews.py
-rw-rw-r-- 1 homer homer 462 Jan 28 15:40 fractals.jdl
drwxrwxr-x 2 homer homer 116 Jan 28 15:40 sandbox
```

Detailed information regarding the operations performed in each of the scripts below is embedded to the comments inside each of the scripts individually.

- Also download the current PiCaS version as *picas.zip* from GitHub and put both PiCaS and the *couchdb.tgz* file in the sandbox directory:

```
$cd sandbox
$curl --location https://github.com/sara-nl/picasclient/archive/master.zip > picas.zip
$mv ../../couchdb.tgz ./
```

- And finally compile the fractals program (and put it in the sandbox directory) and move one directory up again:

```
$cc ../../fractals.c -o fractals -lm
$cd ..
```

The sandbox directory now holds everything we need to send to the Grid worker nodes.

Prepare your Tokens

Create the Tokens

This example includes a bash script (`./createTokens`) that generates a sensible parameter file, with each line representing a set of parameters that the fractals program can be called with. Without arguments it creates a fairly sensible set of 24 lines of parameters. You can generate different sets of parameters by calling the program with a combination of `-q`, `-d` and `-m` arguments, but at the moment no documentation exists on these. We recommend not to use them for the moment.

- After you ran the `createTokens` script you'll see output similar to the following:

```
$. ./createTokens
/tmp/tmp.fZ33Kd8wXK
$cat /tmp/tmp.fZ33Kd8wXK
```

Upload your Tokens to the PiCaS server

Now we will start using PiCaS. For this we need the downloaded CouchDB and PiCaS packages for Python and set the hostname, database name and our credentials for the CouchDB server:

- Edit `sandbox/picasconfig.py` and set the PiCaS host URL, database name, username and password.
- Link the `picasconfig.py` file in the current directory. This makes it available for the scripts that need to upload the tokens to CouchDB:

```
$ln sandbox/picasconfig.py
```

- Make the CouchDB package locally available:

```
$tar -zxf sandbox/couchdb.tgz
```

- Upload the tokens:

```
$python createTokens.py /tmp/tmp.fZ33Kd8wXK
```

- Check your database in this link:

https://nosql01.grid.sara.nl:6984/_utils/database.html?homerdb

replace `homerdb` with your Picas database name

- Create the Views (pools) - independent to the tokens (should be created only once):

```
$python createViews.py
```

Run the example locally

- If you submit the jobs on the UI, the job will start fetching tokens from the pool server and run the application locally on the UI machine:

```
$cd sandbox/
$./startpilot.sh

Connected to the database homerdb sucessfully. Now starting work...
-----
Working on token: token_2
```

```

lock 1453570581
_rev 2-8d7f141114b7335b50612ba4dfb92b3d
hostname ui
exit_code
scrub_count 0
done 0
input -q 0.100 -d 256 -m 8400
output
_id token_2
type token
-----
/usr/bin/time -v ./process_task.sh "-q 0.100 -d 256 -m 8400" token_2 2> logs_token_2.err 1> logs_token_2.out
-----
Working on token: token_6
lock 1453570589
...

```

You can monitor the progress for the Tokens that are waiting, running, finished or in error state, from the PiCaS website here:

https://nosql01.grid.sara.nl:6984/_utils/database.html?homerdb

replace homerdb with your Picas database name

While the UI has started processing tokens, submit the pilot jobs to the Grid. Continue to the next section ...

Run the example on the Grid

- Create a proxy:

```
$startGridSession lsgrid # replace lsgrid with your VO
```

- Submit the pilot jobs:

```
$glite-wms-job-submit -d $USER -o jobIDs fractals.jdl
```

It will recursively generate an image based on parameters received from PiCas. At this point, some of your tokens are processed on the Grid worker nodes and some of the tokens are already processed on the UI. Note that the UI is not meant for production runs, but only for testing few runs before submitting the pilot jobs to the Grid.

- Convert the UI output file to .png format and display the picture:

```
$convert output_token_6 output_token_6.png # replace with your output filename
```

For the tokens that are processed on Grid, you can send the output to the *Grid Storage* or some other remote location.

Checking failed jobs

While your pilot jobs process tasks, you can keep track of their progress through the CouchDB web interface. There are views installed to see:

- all the tasks that still need to be done (Monitor/todo)
- the tasks that are locked (Monitor/locked)
- tasks that encountered errors (Monitor/error)
- tasks that are finished (Monitor/done)

When all your pilot jobs are finished, ideally, you'd want all tasks to be 'done'. However, often you will find that not all jobs finished successfully and some are still in a 'locked' or 'error' state. If this happens, you should investigate what went wrong with these jobs. Incidentally, this will be due to errors with the Grid middleware, network or storage. In those cases, you can remove the locks and submitting some new pilot jobs to try again. In other cases, there could be errors with your task: maybe you've sent the wrong parameters or forgot to download all necessary input files. Reviewing these failed tasks gives you the possibility to correct them and improve your submission scripts. After that, you could run those tasks again, either by removing their locks or by creating new tokens if needed and then submitting new pilot jobs.

5.7 Pilot jobs

In this page we will show you how to run pilot jobs on the Grid and track the status of hundreds jobs running at a time:

Contents

- *Pilot jobs*
 - *About pilot jobs*
 - *Pilot Job Workflow*
 - * *Pilot job database*
 - *Pilot job advantages*
 - *Pilot job submission*
 - *Pilot Job Frameworks*

5.7.1 About pilot jobs

More about Pilot jobs?

See also:

Check out our mooc video [Pilot Job Frameworks](#)

When you have tens, hundreds of jobs that you submit to the Grid you may find yourself writing software which checks the status of all those jobs and tries to find out which ones have succeeded and which ones should be resubmitted.

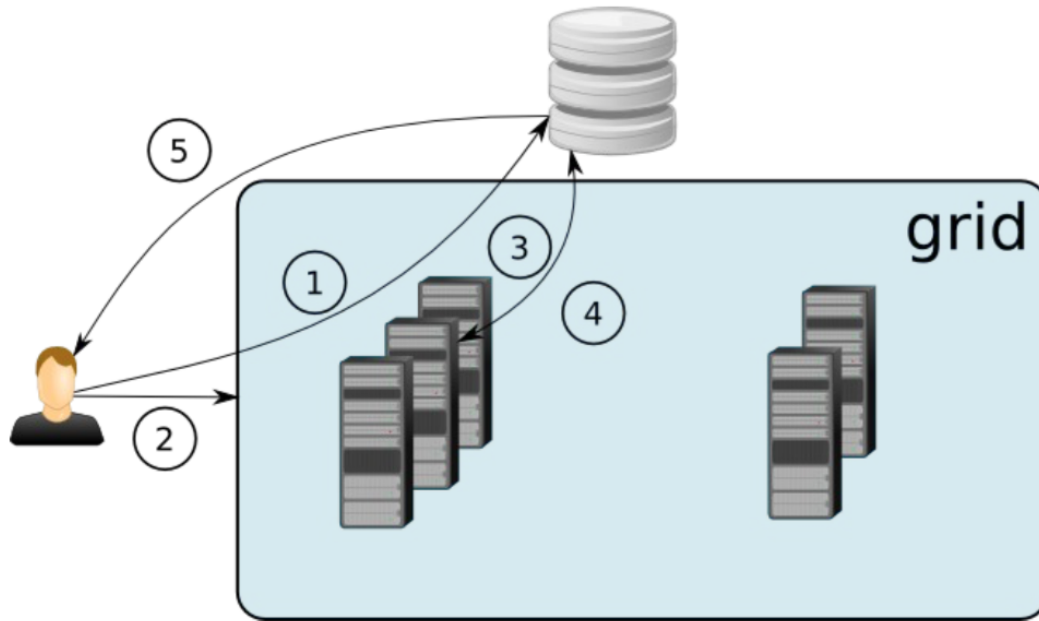
In addition, when submitting a large number of jobs you will often find that they need to operate on the same database or some other static datasource. For every jobsubmission this database then has to be copied to the node the job is running on. This introduces a large overhead per job.

A solution to these problems is to use a pilot job framework. Such frameworks start by submitting a number of pilot jobs to the Grid. Pilot jobs are like normal jobs, but instead of executing the task directly they contact a central server once they are running on a worker node. Then, and only then, will they be assigned a task, get their data and start executing. The central server handles the request from pilot jobs and keeps a log of what tasks are being handled, are finished, and can still be handed out. The pilot job can request a new task, report the successful completion of a task just executed or report that it is still working on the task it received previously. When the task server doesn't hear about the progress or completion of a task it has distributed, it will assume the pilot job is either dead or the job has failed. As a result the task will be assigned to another pilot job after it has made a request for a new task.

A pilot job will not die after it has successfully completed a task, but immediately ask for another one. It will keep asking for new jobs, until there is nothing else to do, or its wall clock time is up. This reduces overhead for jobsubmission considerably.

5.7.2 Pilot Job Workflow

The picture below illustrates the workflow of pilot job systems: (1), the user uploads work to the central database. This could be a list of parameter settings that need to be executed using some application on a computational resource. The user then (2) submits jobs just containing the application to the Grid, which handles retrieving from (3) and updating of (4) the job database. This process continues until all the work present in the database has been done, the application has crashed or the job has run out of time on the computational resource. When all work has been done, the user retrieves (5) the results from the database.



Pilot job database

In *pilot frameworks*, the database server is a server which is not necessarily tied to Grid infrastructure. In fact, you can have simple access to a pilot job server with your Internet browser or with HTTP command line clients like `wget` and `curl`.

The concept of token pools can also be used to create workflows. A task server can deal out tokens from different pools in succession and pilot jobs can remove tokens from pools (by processing them) and create tokens in other pools.

5.7.3 Pilot job advantages

Notice that when you use *pilot frameworks* you do not have to worry about job failures. When jobs fail they will not request and process new tasks. The user should not look at the pilot jobs he/she has submitted but to the number of tasks being processed. There is no need to keep track of submitted (pilot) jobs and resubmit those (and only those) that failed. Remember a pilot job will die if all tasks have been processed.

A second advantage of using pilot jobs is the reduced overhead of job submission. Once a pilot job is in place it will process tasks as long as there are any, or it's wall clock time (the maximum time a job is allowed to run) is up. This can also be advantageous for jobs who all need to transfer a large datafile. In the last example, a database should be downloaded from a SE to the Worker Node where the job is running. Note, that this is always the same database for all jobs. When a pilot job is running and is processing task after task it only needs to download the database once.

5.7.4 Pilot job submission

To be able to submit multiple pilot jobs at once, they are submitted to the Grid with a specific *JDL* type called *Parametric*. Learn more about this technique in *parametric jobs* section.

5.7.5 Pilot Job Frameworks

There are several pilot frameworks for the Grid. At SURFsara we support two of them, *PiCaS* and *ToPoS*:

- *ToPoS Overview*
- *Topos Example*
- *Picas Overview*
- *Picas Example*

5.8 Data replication

In this page we will show an example for running applications on multiple LSG clusters with replicas that also require big volumes data:

Contents

- *Data replication*
 - *Problem description*
 - *Data Requirements*
 - * *Moving output data from the job to the SE*

lcg/lfc/lfn? Only for large files with multiple replicas.

The lectures Data management on the Grid *partI*, *partII* and *partIII* present the *lcg/lfc/lfn Storage clients*. However, we advise you to better use the *globus client* or *srm client*, unless you need to run jobs on multiple sites which require access on the **same** large dataset (or database). In case of doubts, contact us at helpdesk@surfsara.nl.

5.8.1 Problem description

The `InputSandbox` and `OutputSandbox` attributes in the JDL file are the basic way to move files to and from the User Interface (UI) and the Worker Node (WN). However, if large (from about 100 MB and larger) or many files are involved you should not use these sandboxes to move data around. Instead you can use the *Storage Elements* and work with the `lfc` and `lcg` commands. These commands, and the storage system in general, are explained in the section *lcg-lfn-lfc clients*.

Here we give an example of how to use large input and output files which are needed by a job. We will use replicas to avoid transferring e.g. a database to multiple clusters every time.

5.8.2 Data Requirements

This case describes the `DataRequirements` attribute in your job description file; this attribute is a list of classads representing the data requirements for the job. Each classad has to contain three attributes :

InputData The list of input files needed by the job

DataCatalog The type of data catalog - needed by the Grid middleware. This is needed in order to resolve logical names to physical names. Fill in “DLI” here.

DataCatalogType The address (URL) of the data catalog if this is not the VO default one.

The presence of the `DataRequirements` attribute causes the job to run on a Computing Element (CE) which is next to the Storage Element (SE) where the requested file is stored. Note that this attribute doesn’t perform the actual copy of the file from the SE to the WN; as we will see, this has to be done by the user.

To do this, first register a file on a SE and to the LFC. We do this by copy and register (`lcg-cr`):

```
$ lcg-cr --vo lsgrid -d gb-se-ams.els.sara.nl -l lfn:/grid/lsgrid/homer/test.txt file:/home/homer/loc
guid:522350d4-a28a-48aa-939b-d85c9ab5443f
```

Note that the guid part is what we get as return value from the command. It identifies the file uniquely in the Grid storage. You can save this id for emergencies. The part which starts with `lfn:` identifies the logical file name of our uploaded file.

Note: The LFC needs to support your VO in order to work.

Second, create a JDL file that describes your job. It will contain the LFC of the file, as is shown here.

```
$ cat inputdata.jdl
[
    Executable = "/bin/sh";
    Arguments = "scriptInputData.sh lfn:/grid/lsgrid/homer/test.txt";

    StdOutput = "std.out";
    StdError = "std.err";

    InputSandbox = "scriptInputData.sh";
    OutputSandbox = {"std.out", "std.err"};

    DataRequirements = {
        [
            InputData = {"lfn:/grid/lsgrid/homer/test.txt"};
            DataCatalogType = "DLI";
            DataCatalog = "http://lfc.grid.sara.nl:8085";
        ]
    };
    DataAccessProtocol = {"gsiftp"};

    RetryCount = 3;
]
```

This JDL mentions the script `scriptInputData.sh` (as value of `Arguments`) which will be submitted to the WMS, and run on a Worker Node. This script needs an inputfile, and expects an LFN (Logical File Name) as argument. We will use the file that we copied to an SE earlier. In the `DataRequirements` section, we mention the LFN of this file as value of `InputData`. Notice that the `DataCatalogType` and `DataCatalog` are also described. You can copy these values.

Note that this in itself is not enough for the script to use the file. It still needs to be copied to the worker node where the job lands. All that is achieved by this JDL description is that the job will land close to an SE which contains the needed data. The copying is done by the script itself. To actually copy the file associated with this LFN from the SE to the WN (Worker Node), the script uses an `lcg-cp` command. The script `scriptInputData.sh` is shown below.

The script gets the file, performs the `ls` command and shows the content of the file to `stdout`.

```
$ cat scriptInputData.sh
#!/bin/sh

# Set the proper environment
export LFC_HOST=lfc.grid.sara.nl
export LCG_GFAL_INFOSYS=bdii.grid.sara.nl:2170
export LCG_CATALOG_TYPE=lfc

# Download the file from the SE to the WN where this job runs
# note that the LFN is passed as input to this script
lcg-cp --vo lsgrid $1 file:`pwd`/local_file

echo "#####"
ls -la local_file
echo "#####"
# type the file just downloaded
cat local_file
```

Now the actual submission, status checking, output retrieval and inspection can take place. If you want to try this example, you have to create two files, `inputdata.jdl` and `scriptInputData.sh`, filling them with the content displayed above. Of course, you have to register your own file and consequently change the LFN requested within the `DataRequirements` attribute.

Moving output data from the job to the SE

What do you do when you have to move data from a running job on the Worker Node to a Storage Element? The answer is: the job has to do it by having a script copy the data. We give an example. Assume that the following script code is executed by a running job.

```
$ cat registeringfile-script.sh
#!/bin/sh
# Author : Emidio Giorgio
# Usage : register a file to the default SE, with a specified LFN
# - The file to copy and register is passed as first input argument to the script ($1)
# - The logical file name it will have is the second input argument to the script ($2)
# - the LFN will be like this /grid/lsgrid/YOUR_DIRECTORY/$2

# Set the proper environment
export LFC_HOST=lfc.grid.sara.nl
export LCG_GFAL_INFOSYS=bdii.grid.sara.nl:2170
export LCG_CATALOG_TYPE=lfc

# Actually upload the file to the SE
# path to the file to be registered is built as {current path}/{relative path from this script to fi
# REPLACE CHANGEME with an (already existing) LFC directory of your choice
lcg-cr --vo lsgrid -l lfn:/grid/lsgrid/CHANGEME/$2 file:$PWD/$1
```

This script is in charge of copying the output of your job. The simplest thing is to run it from within the main job script, as shown below:


```
$ cat  scriptWhichDoesSomething.sh
#!/bin/sh

# do whatever
echo "This is a very dummy test" > fileout.txt

# run the script which registers the file fileout.txt just created above
/bin/sh registeringfile-script.sh fileout.txt data_from_the_WN

# greetings
echo "All done correctly (I hope). Bye bye"
```

This could be a starting point for your JDL:

```
$ cat  JobWritingToSE.jdl
[
    Executable = "/bin/sh";
    Arguments  = "scriptWhichDoesSomething.sh";

    StdOutput  = "std.out";
    StdError   = "std.err";

    # carry out also the script which registers the file
    InputSandbox = {"scriptWhichDoesSomething.sh", "registeringfile-script.sh"};
    OutputSandbox = {"std.out", "std.err"};
]
```

Alternatively, you can just append the content of `registeringfile-script.sh` to your main script.

Service implementation

6.1 System specifications

This page describes the specifications for the underlying Grid resources, both compute and storage systems:

6.1.1 LSG specifications

This page describes the specifications for the Life Science Grid (LSG) clusters. For a list of the clusters, see *Life Science Grid clusters*.

Contents

- *LSG specifications*
 - *Quick summary*
 - *Worker Nodes*
 - *Storage Node*
 - *Computing Element*
 - *User Interface*
 - *Service Nodes*
 - *Queues*

If you have any questions concerning the technical specifications below, please contact us at helpdesk@surfsara.nl.

Quick summary

Last update: March 2016

LSG cluster	Capacity
Operating system	Linux CentOS 6.x 64bit
Total number of cores	128 CPU cores at 2.3 GHz
Total amount of memory	512 GB
Total scratch space	10TB
Disk storage	40TB of staging area
Network backbone	10Gbit/s local switch, 1Gbit/s external connectivity

Worker Nodes

The jobs will be actually executed on this kind of nodes; this is the hardware specification:

- PowerEdge R815 4xAMD Opteron(tm) Processor 6376
- Type: Bulldozer
- Cores: 64
- RAM: 256GB
- Number of nodes: 2

Storage Node

This node stores all the data related to the user jobs; every user has a default quota of 50GB, unless a specific one is required:

- PowerEdge R515 2xAMD Opteron(tm) Processor 4280
- Type: Bulldozer
- Cores: 16
- RAM: 128 GB
- Number of nodes: 1
- Storage space: 40TB

Computing Element

This is a virtual machine running on one of the Service Nodes; the CE gets the jobs from the users and sends them to the appropriate WNs:

- Cores: 2
- RAM: 4 GB
- Number of nodes: 1

User Interface

This is a virtual machine running on one of the Service Nodes; please be aware that this server's purpose is NOT to execute directly your programs. Use instead Torque commands such as 'qsub' to submit your jobs to the Worker Nodes which have more CPU/Memory capacity. Thank you for your understanding. The VM specifications of the UI are:

- Cores: 8
- RAM: 16 GB
- Number of nodes: 1

Service Nodes

The hardware specifications of the service nodes:

```
* PowerEdge R420 2xIntel(R) Xeon(R) CPU E5-2420 0 @ 1.90GHz
* Type: Sandy Bridge architecture /w Sandy Bridge-EP cores
* Cores: 24
* RAM: 96 GB
* Number of nodes: 2
```

Queues

Queue	Max. Walltime (hh:mm:ss)
express	00:30:00
infra	00:30:00
medium	36:00:00
long	72:00:00

6.1.2 Gina specifications

Gina is the Grid cluster at SURFsara; currently all the servers are located at the Vancis B.V. datacenter divided over two rooms. After different tenders these servers belong to different brands (Dell & Fujitsu). Three service nodes are used to host virtual machines for different purposes (creamces, monitoring, installation, dns servers...). All the worker nodes are installed on physical nodes.

This page describes the specifications for the Gina Grid cluster at SURFsara:

Contents

- *Gina specifications*
 - *Quick summary*
 - *Network*
 - *Worker Nodes*
 - *Service nodes*
 - *CreamCEs*
 - *Queues*

If you have any questions concerning the technical specifications below, please contact us at helpdesk@surfsara.nl.

Quick summary

Gina cluster	Capacity
Operating system	Linux CentOS 6.x 64bit
Total number of cores	5600 Xeon cores at 2.2 to 2.6 GHz
Total memory	41TB
Total scratch space	2100TB
Network backbone	Juniper Q-Fabric Network Fabric which also connects the Grid storage

Last update: January 2016

Network

GinA is connected to a Juniper Q-Fabric network fabric. On this fabric also the Grid storage is connected and makes high throughput possible. Currently we have seen over 20GB/sec (~170-200Gbit/sec) of peak network traffic to the Grid storage. All workernodes are connected with a single 10Gbit ethernet connection.

Worker Nodes

This is the list of the different worker nodes in order of installation/configuration from newest to oldest:

Worker Nodes am90-{01-33}, am91-{01-33}, am94-{01-33}:

```
* Dell R630
* 2x Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz (24 cores)
* 192GB RAM
* ~8000GB scratch
* Type: Haswell architecture /w Haswell-EP cores (2014)
* Number of nodes: 100
* Total cores of this type: 2400
* Scratch per core: ~300GB
* RAM per core: 8GB
```

Worker Nodes ar90-{01-52}, ar91-{01-52}:

```
* Fujitsu CX250
* 2x Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60GHz (16 cores)
* 128GB RAM
* ~11TB scratch
* Type: Ivy Bridge architecture /w Ivy Bridge-EP cores (2013)
* Number of nodes: 104
* Total cores of this type: 1664
* Scratch per core: ~680GB
* RAM per core: 8GB
```

Worker Nodes v37-{01-12}, v33-{01-06}:

```
* Dell R820
* 4x Intel(R) Xeon(R) CPU E5-4620 0 @ 2.20GHz (32 cores)
* 256GB RAM
* 6.8TiB scratch
* Type: Sandy Bridge architecture /w Sandy Bridge-EP cores (2012)
* Number of nodes: 18
* Total cores of this type: 576
* Scratch per core: ~200GB
* RAM per core: 8GB
```

Worker Nodes am95-{01-48}, v33-{17-48}:

```
* Dell M610
* 2x Intel(R) Xeon(R) CPU E5649 @ 2.53GHz (12 cores)
* 48GB RAM
* ~850 GB scratch
* Type: Nehalem architecture /w Westmere-EP cores (2011)
* Number of nodes: 80
* Total cores of this type: 960
* Scratch per core: ~70GB
* RAM per core: 4GB
```

Service nodes

Service{01,02,03}:

```
* Dell R420
* 2xIntel(R) Xeon(R) CPU E5-2420 0 @ 1.90GHz (12 cores)
* 96GB RAM
* Number of nodes: 3
* RAM per core: 8GB
```

CreamCEs

All 3 CreamCEs are virtualized and distributed among the 3 Service Nodes. Every CreamCE has 4 cores and 9GB RAM in total.

Queues

Queue	Max. CPU Time (hh:mm:ss)	Max. Walltime (hh:mm:ss)	VOs(group) allowed
ex-treme	120:00:00	120:00:00	emutd
long	96:00:00	96:00:00	geant4 atlas(production)
medi-umc*	n/a	72:00:00	bbmri.nl(RP2) lofar lsgrid(mediummc) projects.nl(geomodel) pvier
medium	36:00:00	36:00:00	astron atlas alice bbmri.nl beapps biomed dans drihm.eu enmr.eu esr euclid-ec.org geant4 lhcb lofar lsgrid nlesc.nl omegac pvier vlemmed xenon.biggrid.nl projects.nl
medium-8gb**	36:00:00	36:00:00	alice
short	04:00:00	04:00:00	astron atlas bbmri.nl beapps biomed dans drihm.eu enmr.eu esr euclid-ec.org geant4 lhcb lofar lsgrid nlesc.nl omegac pvier vlemmed xenon.biggrid.nl projects.nl
infra	02:00:00	00:30:00	dteam ops pvier

* This is a queue for multicore jobs

** This is a queue for jobs that require 8GB per core

6.1.3 dCache specifications

This page describes the technical specifications for dCache. If you have any questions concerning the technical specifications below, please contact us at helpdesk@surfsara.nl.

Contents

- *dCache specifications*
 - *About*
 - *Disk storage*
 - *Tape storage*
 - *Transfer performance*
 - * *Bandwidth*
 - * *Limits*
 - *Number of transfers per pool*
 - *A single SRM door*

About

dCache has the following concepts:

- A `pool` is a location on a server that can contain files.
- A `poolgroup` is a group of similar pools, assigned to a user group. Usually the pools are on different nodes to distribute the load.
- Our ~60 pool nodes are also configured as doors.
- A `door` is a service that you can contact to send, receive or delete files or restore files from tape using a specific protocol. Protocols are:
 - GridFTP (port range 20000-25000)
 - WebDAV
 - xroot
 - GSIdCap
 - dCache also offers NFS doors, but we currently don't support it; for special cases we might consider supporting this protocol.
- The `namespace` contains a list of all files and their metadata. The files are structured in a virtual directory structure, starting with `/pnfs/grid.sara.nl/`. Each directory can be mapped onto a pool group. Subdirectories inherit this mapping from their parent directory.

Here's a list of accessible dCache nodes:

- `srm.grid.sara.nl`
- `dcmain.grid.sara.nl:2288` (a dCache web interface showing detailed configuration information)
- pool nodes:

```
* fly{1..10}.grid.sara.nl
* bw27-{1..9}.grid.sara.nl
* bw32-{1..9}.grid.sara.nl
* by27-{1..9}.grid.sara.nl
* by32-{1..9}.grid.sara.nl
* rabbit{1..3}.grid.sara.nl
* v40-{8..10}.grid.sara.nl
* whale{1..6}.grid.sara.nl
```

We have these DNS round robin aliases pointing to our doors:

- `gridftp.grid.sara.nl`

- webdav.grid.sara.nl

Here are some metrics per user group: web.grid.sara.nl/dcache.php

The subnet is 145.100.32.0/22. You may need to change your firewall to access this subnet.

Disk storage

We currently (January 2016) have ~8 petabyte of disk storage capacity. This space is divided over several pool groups. These pool groups are for disk only data, t1d1 data (disk data with a tape replica) and for online caching of tape only data.

Tape storage

The Grid tape storage back-end contains ~22 petabyte of data (January 2016). There are two tape libraries: one in Almere and one in the Vancis datacenter in Amsterdam Science Park. Some data only has a single copy, but smaller projects typically have a double copy of their tape data.

Transfer performance

Bandwidth

With dCache, we have reached bandwidths up to 25 gigabyte/s, and dCache is probably capable of much more, depending on the circumstances. This bandwidth was reached between the Gina compute cluster and the dCache cluster. Once, during an internal data migration operation, we have transferred 1 petabyte in 24 hours. With external endpoints however, bandwidth is most likely limited by the network path.

Limits

Number of transfers per pool Each pool supports up to a certain number of concurrent transfers. The specific number for a certain pool group can be looked up in the [dCache web interface](#). If the limit is reached, transfers will be queued, and they will seem stalled. After some time, transfers may time out. But even if they don't, your job may waste valuable computing time waiting for input files that don't arrive.

If that happens, you should reduce your number of concurrent transfers, or ask us whether the limit can be increased. We can not increase the limit endlessly because this would make our systems unstable. Not sure how to proceed? We can help! Contact us at helpdesk@surfsara.nl.

A single SRM door Also the SRM door has some limitations. There's only one of that kind, and sometimes it might be a performance bottleneck. It may be wise to bypass the SRM door and use GridFTP and WebDAV doors directly when possible. If in doubt, feel free to contact us for advice.

6.1.4 Grid User Interface machine

The Grid User Interface (UI) machine is a virtual machine. Its specifications are given below. The hardware specifications of the host machine are also included.

Grid UI machine:

```
* Virtual machine
* Cores: 4
* RAM: 16 GB
```

Host machine:

```
* Poweredge R630
* CPU: Intel Xeon E5645 @ 2.40GHz
* Number of CPU's: 2
* Number of cores per CPU: 6
* Total number of cores: 12
* RAM: 48 GB
* Network: 10 Gb/s
```

If you have any questions concerning these technical specifications, please contact us at helpdesk@surfsara.nl.

If you have any questions concerning the technical specifications of any of our Grid systems, please contact us at helpdesk@surfsara.nl.

6.2 Downtimes and maintenances

Here you will find details to keep informed for the ongoing and upcoming downtimes and maintenances in our Grid systems:

Contents

- *Downtimes and maintenances*
 - *Ongoing maintenances*
 - *Upcoming maintenances*

6.2.1 Ongoing maintenances

You can lookup announcements for the current status of the Grid systems, including all ongoing downtimes and maintenances on [our website user info](#) under:

Systems > System status > National e-Infrastructure Grid Maintenances

Direct link: <http://web.grid.sara.nl/cgi-bin/eInfra.py>

6.2.2 Upcoming maintenances

If you use want to receive notifications and be prepared for upcoming downtimes and maintenances, you can create personal subscriptions to receive announcements tailored specifically for the Grid services and clusters that are relevant for you.

These subscriptions can be created here: <https://operations-portal.egi.eu/dns>.

Note: You can use this service only if you have your Grid certificate installed in you browser.

If you need help creating your subscription(s) we are of course more than willing to assist at helpdesk@surfsara.nl.

6.3 Statistics and monitoring

In this section you will find information about the Grid systems status and usage statistics:

Contents

- *Statistics and monitoring*
 - *SURFsara systems*
 - *NIKHEF facility*
 - *EGI accounting and operations portals*

6.3.1 SURFsara systems

The webpage below contains monitoring information about the usage of various systems at SURFsara:

- [SURFsara systems usage](#)

More specific monitoring information about the Grid systems can be found in the following website:

- [SURFsara Grid systems](#)

6.3.2 NIKHEF facility

The webpage below contains Nikhef facility statistics and status overview:

- [Statistics NIKHEF facility](#)

6.3.3 EGI accounting and operations portals

The portal below provides information about accounting statistics for sites, regions, VOs:

- [EGI Accounting portal](#)

The following site provides Grid sites info updated by participating NGIs:

- [GOCDB NGIs info](#)

7.1 MOOC - Introduction to Grid Computing

This page includes the complete course material as presented during the SURFsara MOOC Introduction to Grid Computing:

Contents

- *MOOC - Introduction to Grid Computing*
 - *About*
 - *Animations*
 - *Lectures*
 - *Use cases*
 - *EGI-InSPIRE*

7.1.1 About

As of 18 November 2013, SURFsara offered the Massive Open Online Course (MOOC) Introduction to Grid Computing. The course aimed to help researchers to understand the Grid key concepts and the role of Grid Computing in computationally intensive problems. This included working with portals, workflow management systems and the Grid middleware.

The taught material consists of:

- a set of **video lectures**.
- a set of **quizzes, assignments** and **exercises**.
- real world examples of **use cases** on Grid.

Note: The mooc participants were provided with student accounts and a preconfigured Virtual Machine (VM) with all the necessary Grid tools installed. If you want to run the examples presented in the video lectures below you will have to request your personal Grid account, see [Prerequisites](#). Please contact helpdesk@surfsara.nl if you need help with this.

7.1.2 Animations

We have prepared a set of animations to display the basic usage of Grid Infrastructure. The animations were presented during the lectures, but they are still accessible to try out freely in this link: [Mooc Animations](#).

7.1.3 Lectures

Course Overview

Course Overview slides pdf

Introduction to Parallel and Distributed Computing

Intro to Parallel and Distributed Computing pdf

Cluster Computing

Cluster Computing pdf

Grid Computing Overview

Grid Computing Overview pdf

Grid_Glossary

Grid Glossary pdf

Hands-on set

Exercises Distributed Cluster Grid pdf

Quiz Distributed Cluster Grid pdf

Working Environment - Grid prerequisites

Working Environment_I Grid prerequisites pdf

Working Environment - Remote access

Working Environment II Remote access pdf

Code gridpi tar

Grid Certificate - Security

Grid Certificate I (security) pdf

Grid Certificate I (extras) pdf

Obtain a Grid Certificate

DigiCert: new way to request a certificate!

Note: DigiCert CA allows you to get your Grid certificate instantly from the GEANT Trusted Certificate Service (instead of Terena portal), by using your institutional login and SURFconext. Read the [User Guide](#) or login directly on the [DigiCert portal](#).

Grid Certificate II (Obtaining a certificate) pdf

User Interface machine

User Interface machine pdf

Virtual Organisations

Virtual Organisations pdf

Hands-on set

Exercises Install Certificate pdf

Quiz_Install Certificate pdf

Grid job Lifecycle

Grid job Lifecycle pdf

Start a Grid_Session

Start a Grid Session pdf

My First Grid job

My First Grid job pdf

Code MyFirstJob tar

Grid Toolkit

Grid Toolkit pdf

Hands-on set

Quick start guide pdf

Exercises First Grid job pdf

Quiz_First Grid job pdf

Application_submission to Grid I

Application submission to Grid I script pdf

Code script tar

Application_submission to Grid II

Application submission to Grid II executable pdf

Code compiled tar

Advanced Grid jobs I

Advanced Grid jobs I Collections & Parametric pdf

Code Collections Parametric tar

Advanced Grid jobs II

Advanced Grid jobs II Multicore pdf

Code multicore tar

Data parallel processing Hadoop

Data parallel processing Hadoop pdf

Hands-on set

Exercises Advanced Jobs pdf

Quiz Advanced Jobs pdf

lcg/lfc/lfn? Only for large files with multiple replicas.

The lectures Data Management on the Grid [1–3] present the lcg/lfc/lfn *Storage clients*. However, we advise you to better use the *globus client* or *srm client* tools, unless you need to run jobs on multiple sites which require access on the **same** large dataset (or database). In case of doubts, contact us at helpdesk@surfsara.nl.

Data Management on the Grid I

Data Management on the Grid I pdf

Data Management on the Grid II

Data Management on the Grid II pdf

Data Management on the Grid III

Data Management on the Grid III pdf

Code DMLargefiles tar

Storage Resource Manager

Storage Resource manager pdf

Code DMsrm tar

Hands-on set

Exercises_Data Management pdf

Quiz_Data Management pdf

Introduction to Workflows I

Introduction to Workflows I pdf

Introduction to Workflows II

Introduction to Workflows II pdf

WS-Pgrade I

WSpgrade I pdf

WS-Pgrade II

WSpgrade II pdf

Science Gateways

Science Gateways pdf

Hands-on set

Exercises Workflows pdf

Code Exercises Workflows tar

Code Solutions Workflows tar, [W6 screencast1](#), [W6 screencast2](#), [W6 screencast3](#), [W6 screencast4](#), [W6 screencast5](#), [W6 screencast6](#), [W6 screencast7](#).

Quiz Workflows pdf

Pilot Job Frameworks

Pilot job frameworks pdf

Picas Server side I

Picas server side I pdf

Picas Server side II

Picas server side II pdf

Picas client side

Picas client side pdf

Picas practise I

Code Picas tar

Picas practise II

Code Picas tar

Course Summary

Course summary pdf

7.1.4 Use cases

Use case: EGI overview

EGI overview pdf

Use case: LOFAR

Extreme physics in space pdf

Use case: Climate change over Europe

Climate change pdf

Use case: VisIVO Science Gateway

VisIVO Science Gateway pdf

Use case: Picas

Picas pdf

Use case: Molecular Biology

Molecular Biology pdf

7.1.5 EGI-InSPIRE

The work is supported by the EGI-InSPIRE project (Integrated Sustainable Pan-European Infrastructure for Researchers in Europe), co-funded by the European Commission (contract number: RI-261323) for four years from the 1st of May 2010. EGI-InSPIRE is a collaborative effort involving more than 50 institutions in over 40 countries. Its mission is to establish a sustainable European Grid Infrastructure (EGI).

- *MOOC - Introduction to Grid Computing*
- *gLite tutorial 2008*

Frequently asked questions

8.1 Frequently asked questions

Check out in this page the most commonly asked questions about Grid. If you still have questions, please contact us at helpdesk@surfsara.nl:

Contents

- *Frequently asked questions*
 - *Getting started*
 - * *I never worked with the Grid before. Where is a good starting point?*
 - * *Where can I lookup up Grid terms?*
 - *Certificates*
 - * *How can I change my Grid certificate password?*
 - * *Unable to load certificate error*
 - * *What are the correct permissions for my certificate files?*
 - * *Couldn't find valid credentials error*
 - * *Get non-vomsified proxy locally*
 - * *How can I renew my certificate?*
 - * *Does my key match the certificate?*
 - * *What is the expiry date of my certificate?*
 - * *How can I see the subject of my certificate?*
 - *Using resources*
 - * *How many cpu's, nodes does the Grid offer?*
 - * *How many cpu hours are available?*
 - * *What is the average amount of memory available per node?*
 - * *What is the data transfer speed between Grid locations?*
 - * *How can I calculate the total CPU time I consumed?*
 - * *System usage and CPU efficiency*
 - * *How can I find all the available LSG Storage Elements and get their SURLS?*
 - * *How can I find all the available LSG Compute Elements and use in my JDL?*
 - * *Do I need to switch from my local LSG cluster to Grid?*
 - * *How to run PBS jobs with wallclock greater than 36 hours on LSG?*
 - *Troubleshooting*
 - * *General troubleshooting steps*
 - * *How can I get more logging info for my job?*
 - * *File transfers don't start*

8.1.1 Getting started

I never worked with the Grid before. Where is a good starting point?

New users are advised to read through our tutorial page, the *Prerequisites* and *First Grid job* which guides you through the whole process from getting a Grid certificate to the actual job run. The Grid team at SURFsara is willing to assist, just contact helpdesk@surfsara.nl.

Where can I lookup up Grid terms?

Check out the file `Grid Glossary pdf` that contains most of the basic Grid terminology and abbreviations.

8.1.2 Certificates

How can I change my Grid certificate password?

Before you create a new private key file with a new password, we recommend you to make a backup of the old `userkey.pem` file.

To change your Grid certificate password, type:

```
$openssl rsa -in ~/.globus/userkey.pem -des3 -out ~/.globus/new_private_key_file
$mv ~/.globus/new_private_key_file ~/.globus/userkey.pem # this will replace your old key file with
```

Note: this only changes the password you use for your certificate. If you think your certificate is compromised, you HAVE to revoke your certificate!

Unable to load certificate error

If you get the following error:

```
unable to load certificate 17714:error:0906D064:PEM routines:PEM_read_bio:bad base64
decode:pem_lib.c:781:
```

when you use the command `openssl x509 -text -noout -in usercert.pem`, it means that the email with the certificate wasn't saved properly as plain text (it included the Mime type for formatting). Repeat carefully the steps as described in *Retrieve your DutchGrid certificate* section.

What are the correct permissions for my certificate files?

- Set the proper permissions to your certificate files:

```
$chmod 644 usercert.pem
$chmod 400 userkey.pem
```

- Verify the correct permissions:

```
$cd $HOME/.globus
$ls -l

-rw-r--r--      1 homer    homer          4499  May 10 13:47  usercert.pem
-r-----      1 homer    homer           963  May 10 13:43  userkey.pem
```

Note that the private key file should be **read-only** and only readable to you.

Couldn't find valid credentials error

If you get the following error when creating a new proxy:

```
ERROR: Couldn't find valid credentials to generate a proxy.
Use --debug for further information.
```

The permissions on your installed certificate are probably wrong. Set the *correct permissions* and try creating a proxy again.

Get non-vomsified proxy locally

- To download locally the proxy stored on *MyProxy server* you need to set a passphrase upon creation. To do this, protect your proxy with a MyProxy pass phrase by omitting option “-n”:

```
$myproxy-init -d
```

It will first ask your Grid certificate password and then prompt you to enter a MyProxy passphrase twice. You will use the latter passphrase to download your proxy.

Here is an example of the displayed output:

```
Your identity: /O=dutchgrid/O=users/O=sara/CN=Homer Simpson
Enter GRID pass phrase for this identity:
Creating proxy ..... Done
Proxy Verify OK
Your proxy is valid until: Wed Jan 13 14:35:00 2016
Enter MyProxy pass phrase:
Verifying - Enter MyProxy pass phrase:
A proxy valid for 168 hours (7.0 days) for user /O=dutchgrid/O=users/O=sara/CN=Homer Simpson now
```

- Now use the MyProxy pass phrase to get this proxy locally on the UI:

```
$myproxy-get-delegation -d
```

Here is an example of the displayed output:

```
Enter MyProxy pass phrase:
A credential has been received for user /O=dutchgrid/O=users/O=sara/CN=Homer Simpson in /tmp/x50
```

Note that the downloaded proxy will not include the voms attributes.

How can I renew my certificate?

The personal Grid certificates are valid for a year. This means that every year you need to renew your personal Grid certificate. The procedure for renewing your certificate depends on your CA, either DigiCert or DutchGrid.

- For *DigiCert* Grid certificate, you can request a new certificate anytime from the [DigiCert portal](#). Follow this guide to *obtain and install a DigiCert Grid certificate*.
- For *DutchGrid* Grid certificate, you have two options:
 - When your certificate has already expired, you *have* to request a new certificate from scratch with the jGridstart tool. Follow this guide to *obtain a DutchGrid certificate*.
 - If your current certificate has *not* expired yet, you can *renew* your certificate. This is a faster procedure because you avoid revisiting your RA for your id verification. What you need to do:
 1. Log in to the UI with X session enabled.

2. Start the jGridstart tool on the UI (assuming that your current certificate is installed there): `java -jar jgridstart-wrapper-XX.jar`
3. Select **Actions** -> **Renew** from the menu bar.
4. Generate a new request by verifying your details (name, surname, email, organisation). At this stage you will provide a new password for your new Grid certificate - make sure you keep this safe! Click “Next”.
5. Submit the request. This will create a new private `userkey.pem` file in your `~/ .globus` directory. Click “Next”.
6. You will receive your new certificate within few days via email. Once received, follow the instructions to *install it on the UI*.

Keep in mind that when you renew your certificate the certificate key will change too. To avoid mixing up the old and new certificate files, check whether your new certificate and key *match each other*.

Does my key match the certificate?

Using the modulus you can see whether a key and a certificate match. The modulus is a short message which can be used to identify a private key and the key which was signed with the certificate. If they match, the certificate signs that private key. If not, you may have mixed up different key or certificate files.

To find the modulus of your key, use:

```
$openssl rsa -in userkey.pem -noout -modulus
```

which requires the key which you used to protect your key file. To find the modulus of your certificate, use:

```
$openssl x509 -in usercert.pem -noout -modulus
```

If the moduli of the key file and the certificate file do not match, you cannot use that combination to identify yourself.

What is the expiry date of my certificate?

To find out when your certificate is valid, use:

```
$openssl x509 -in usercert.pem -noout -dates
```

This will tell you when your certificate is valid.

Note that a key does not have a validity period.

How can I see the subject of my certificate?

The subject of a certificate is the human-readable identification of who the certificate belongs to. It usually contains your name, country, organisation and your e-mail address.

To find out who the certificate belongs to, use:

```
$openssl x509 -in usercert.pem -noout -subject
```


8.1.3 Using resources

How many cpu's, nodes does the Grid offer?

The Grid infrastructure is interconnected clusters in Netherlands and abroad. The users can get access to multiple of these clusters based on their *Virtual Organisation*.

- Global picture: 170 datacenters in 36 countries: in total more than 330000 compute cores, 500 PB disk, 500 PB tape.
- In the Netherlands NGI_NL infrastructure: 14 datacenters (3 large Grid clusters, 11 smaller ones): in total approximately 10000 compute cores, 12 PB disk, tape capacity up to 170 PB.

How many cpu hours are available?

The available core hours and storage depend on the funding models. We make tailored agreements to incorporate the user requirements and grant resources based on the applicable funding scheme.

What is the average amount of memory available per node?

The average memory per node depends on number of cores per node. It is typically 8GB per core, but the nodes vary between 12 and 64 cores per node (48 to 256GB RAM per node).

What is the data transfer speed between Grid locations?

In the Netherlands NGI_NL infrastructure the transfer speed between Grid storage and Grid processing cluster (at SURFsara) is up to 500Gbit/s. The transfer speed between nodes is 10Gbit/s and between sites it is typically 10 to 20 Gbit/s.

How can I calculate the total CPU time I consumed?

The total CPU time depends on the amount of cores that your application is using and the wallclock time that the corresponding job takes to finish:

```
CPU time = #cores x wallclock(per job) x #jobs
```

For example, let's say that a single job takes 12 h to finish on a 4-core machine and we submitted 10,000 of those. The total CPU time spent is:

```
CPU time = 4cores x 12h x 10,000 = 480,000 CPU hours ~ 55 CPU years
```

System usage and CPU efficiency

CPU efficiency is an important factor to detect if the jobs run smoothly on the infrastructure. The CPU efficiency depends on the real CPU usage and the WallClock time for the job to finish:

```
CPU efficiency = CPU time / WallClock time
```

If the CPU was efficiently being used during the job runtime, then a single core job will have efficiency close to 100%. For multicore jobs the efficiency is higher than 100%.

How can I find all the available LSG Storage Elements and get their SURLS?

- To find out the available SEs for a certain VO, type:

```
$lcg-infosites --vo lsgrid se
```

- To specify a specific SURL (Storage URL), use the following syntax:

```
srn://gb-se-amc.amc.nl:8446/dpm/amc.nl/home/lsgrid/ # storage element at AMC
```

A complete list of the LSG SURLs can be found at [life-science-clusters#cluster-details](#)

How can I find all the available LSG Compute Elements and use in my JDL?

- To find out the available CEs (Compute Elements) for a certain VO, type:

```
$lcg-infosites --vo lsgrid ce
```

Note here that the Total, Running and Waiting numbers are per queue, and the CPU and Free number are per cluster.

- To specify a specific cluster in your JDL file, use the following syntax:

```
Requirements = (RegExp("rug",other.GlueCEUniqueID)); # this requires the job to land on the "rug"
# or you can specify the full UI hostname
Requirements = RegExp("gb-ce-lumc.lumc.nl",other.GlueCEUniqueID); # job lands at lumc
```

Do I need to switch from my local LSG cluster to Grid?

If your local cluster is too busy to get a priority or if you want to run hundreds of jobs at the same time, then we advise you to submit through the Grid middleware instead of submitting to the queue directly. There is obviously more capacity when you scale out to multiple clusters and even if there is maintenance on one cluster, your jobs will then be scheduled on other clusters.

How to run PBS jobs with wallclock greater than 36 hours on LSG?

In order to run PBS jobs on the LSG that last more than 36 hours, you need to *select the proper queue* with the `-q` flag in your `qsub` command when submitting the job:

- If you do *not* use `-q` flag and `lwalltime` directive, then the medium queue is picked and jobs lasting more than 36 hours will be killed.
- If you do *not* use `-q` flag but specify `-lwalltime` directive with value larger than 36 hours, then you request more walltime than the max walltime available in the default medium queue and the job does not start at all.
- If you use the `-q` flag, it is sufficient to get your jobs running for the amount of hours that the specified queue permits.

8.1.4 Troubleshooting

General troubleshooting steps

Don't hesitate to contact us when things go wrong! We're happy to help you overcome the difficulties that every Grid user faces.

In order to assist you better, we have a few troubleshooting steps that may already get you going and otherwise may help us to help you.

- Check the output of `voms-proxy-info -all`. Is your proxy still valid? Does it have the correct attributes for the work you're doing?
- Try running your command with higher debugging level or verbosity.

```
$glite-wms-job-submit --debug ...
$rmcp -debug ...
$gfal-copy --verbose ...
$globus-url-copy -debugftp -verbose-perf -verbose ...
$curl --verbose ...
```

- Is the resource you're using in downtime? Downtimes are announced in the [GOCDB \(Grid Operations Center Database\)](#) (certificate in your browser required). There is also a [list of downtimes of the Dutch Grid sites](#).
- Can you connect to the service?

```
## A basic firewall check: can you connect to the port?
$telnet srm.grid.sara.nl 8443

## Testing the SSL layer of a connection to the dCache SRM door
$echo 'QUIT' | openssl s_client -connect srm.grid.sara.nl:8443 \
    -CApath /etc/grid-security/certificates
## One of the last lines should be: 'Verify return code: 0 (ok)'

## Testing a gridFTP door, control channel
$telnet rabbit1.grid.sara.nl 2811

## GridFTP data channels are more difficult to test, because the port opens only after a transfer
## But after we start an iperf service, you can try to telnet to it.
$telnet rabbit1.grid.sara.nl 24000

## Or just test with iperf:
$iperf3 -c rabbit1.grid.sara.nl -p 24000
## Keep in mind that we have to start iperf first!
```

How can I get more logging info for my job?

To find out more info about the status of your job, use:

```
$glite-wms-job-logging-info -v 2 https://wms2.grid.sara.nl:9000/PHyeyedClEYBjP9l_Xq9mQ # replace with your URL
```

And if you use a file to store your jobs, run:

```
$glite-wms-job-logging-info -v 2 -i jobIds # replace jobIds with your file
```

File transfers don't start

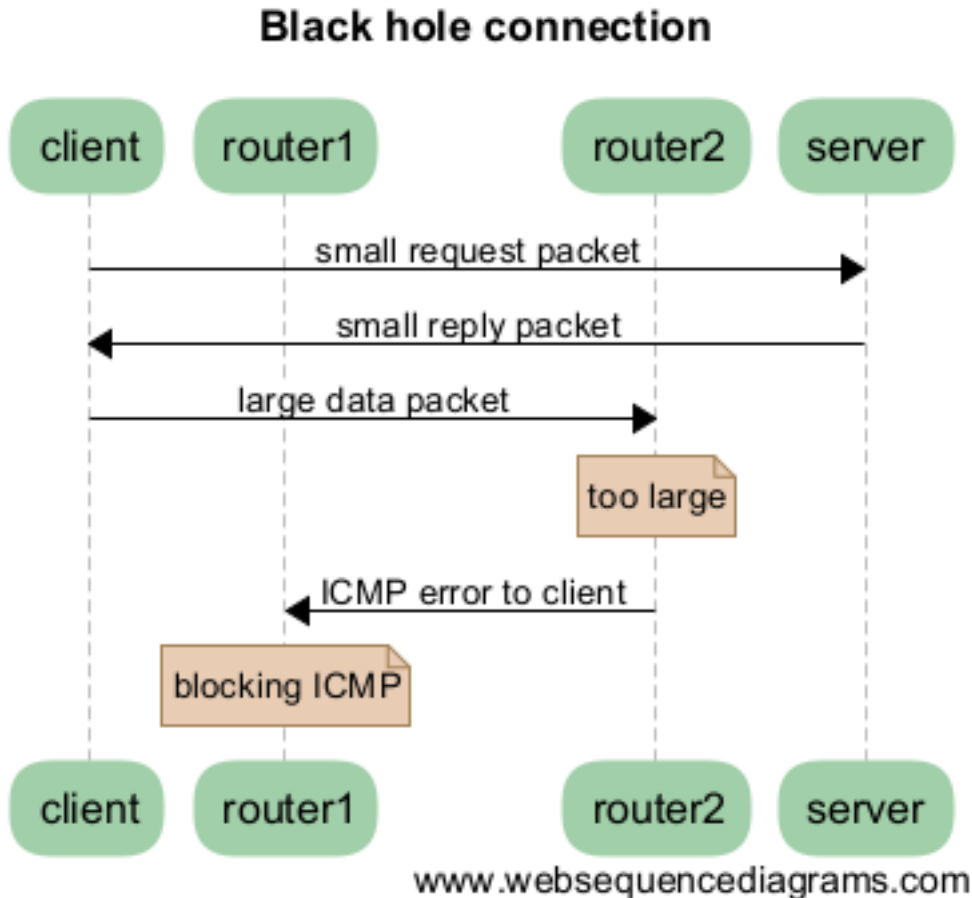
Occasionally, transfers are stuck when 0 bytes have been transferred. There are some common causes for stalled transfers.

- A firewall blocks the ports for the data channel. If you use `rmcp`, specify `--server_mode=passive`. If that doesn't help, check whether your firewall allows outgoing traffic to ports 20000 to 25000 (GridFTP data channel range).

- You've reached the maximum number of transfers for the storage pools that have been allocated to you. All transfers beyond the maximum will be queued, until previous transfers finish to make 'transfer slots' available. This could mean that some of your jobs are wasting CPU time while they wait for input files. This is not efficient. It's better to reduce the number of concurrent transfers so that you don't reach the maximum.

You can see whether this happens at [these graphs](#). A red color ('Movers queued') means that there are stalling transfers.

- You're transferring files from/to outside SURFsara, and your endpoint support a MTU (network packet) size of 9000, but the network path doesn't. Control traffic passes through because it consists of small packets. But data traffic consists of large packets and these are blocked. The image below illustrates this:



Some tools to test this:

```
# Run this from your endpoint of the transfer; adjust the value to find the limit.
# Check first whether your system supports a MTU of 9000.
ping -M do -s 8972 gridftp.grid.sara.nl

# This command tells you what the supported MTU value is.
tracert gridftp.grid.sara.nl
```

Another good tool for testing the network is iperf. We'll start an iperf server at your request so that you can test against it.

```
$iperf -c rabbit1.grid.sara.nl --port 24000 --parallel 4
```

A fix for Linux servers is to enable `tcp_mtu_probing` in `sysctl.conf`. This enables the Linux kernel to select the best MTU value for a certain network route.

8.2 Cheatsheet

Look at the raw version of this file to compare the source and build version: `sphinx_cheatsheet.rst`

The first lines of this page will also be explained later on in this document (see *Links* and *Titles*)

italic word

bold word

This is text in Courier

This is plain text.

- This is a bulleted list.
- ...
- 1. This is a numbered list.
- 2. It has two items.
 - this is
 - a list
 - with a nested list
 - and some subitems
- and here the parent list continues

Python

or

[SURFsara website](#) (see bottom of the document; that is were we tell Sphinx were SURFsara website should point to)

This is an implicit link to title:

Sample H2

Internal wiki link:

Reference tag: place above a title: `.. _my-reference-label:`

Then refer to it from another page as. For example, for this cheatsheet: `cheatsheet` or `ref:other label <cheatsheet>`

This is a command:

```
openssl pkcs12 -in user.p12 -out userkey.pem -nocerts
```

And this is a code block:

```
$openssl pkcs12 -in user.p12 -out userkey.pem -nocerts
```

8.3 H1: document title

8.3.1 Sample H2

Sample H3

Sample H4

Sample H5

Sample H6 And some text.

Header 1	Header 2	Header 3
body row 1	column 2	column 3
body row 2	Cells may span columns.	
body row 3	Cells may span rows.	
body row 4		

or

Column1	Column2
value1	40
value2	41
value3	42

Note: This is a **note** box.

Warning: This is a **warning** box.

See also:

This is a simple **seealso** note.

Your Topic Title

Subsequent indented lines comprise the body of the topic, and are interpreted as body elements.

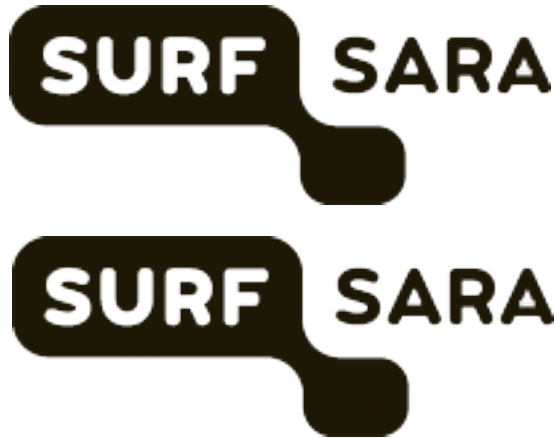
Sidebar Title

Optional Sidebar Subtitle

Subsequent indented lines comprise the body of the sidebar, and are interpreted as body elements.

path/to/myfile.txt

A file for download



8.4 Setup readthedocs Docker image

8.4.1 Preparation

Before you start with the Docker image, make sure that:

- You have [Docker Engine](#) installed
- Docker is running on your host
- You have cloned the `griddocs` repository

8.4.2 Build image

The Dockerfile for this image can be found in ReadTheDoc's GitHub repository: <https://github.com/rtfd/readthedocs-docker-images> It is also added as a submodule.

To build the image in Linux, run:

```
1 $git submodule init
2 $git submodule update
3 $cd readthedocs-docker-images
4 $sudo docker build -t rtfd-build:base base/
```

Note: In Mac OS X (with `boot2docker` or `docker-machine`), you can use the same command (`docker build`), except you do not need to prefix the build line with `sudo`.

8.5 Sphinx-install

8.5.1 Introduction

The Grid documentation itself is written in *restructured text*. The HTML documentation is generated using *Sphinx*. This page explains how to install the Sphinx software.

8.5.2 Installing Sphinx

Linux

On recent CentOS distributions, *Sphinx* is included in [epel](#). Fedora has it included in the standard distribution. Install *sphinx* with:

```
sudo yum install python-sphinx
```

On Ubuntu, install with:

```
sudo apt-get install python-sphinx
```

Mac OS X

On Mac OS X, you will need [macports](#), so get that if you have not done so.

Next, follow the instructions on the [sphinx installation page](#).

Warning: the standard terminal in Mac OS X has locale settings incompatible with sphinx; to change it once for a terminal session, type:

```
export -n LC_CTYPE
```

To change it permanently:

1. open the terminal preferences
2. go to the tab ‘advanced’
3. uncheck “Set locale environment variables on startup”

8.6 Documentation how-to

This page explains how you can edit the documentation, how to generate HTML documentation from the source and how to install the software needed for this. Some handy links:

- The Grid documentation page is: <http://doc.grid.surfsara.nl/>
- The source code of documentation is in GitHub: <https://github.com/sara-nl/griddocs/tree/master/source/>
- The readthedocs overview page is: <https://readthedocs.org/projects/grid-documentation/>

When changes are committed using Git and pushed to the SURFsara GitHub repository, the documentation is automatically rebuild and published by readthedocs.org.

We have some style guidelines to encourage a consistent style. But don’t worry too much about that: any contribution is welcome.

8.6.1 Contributing through GitHub

You are welcome to add and improve the documentation directly to the repository. For this, you’ll need a GitHub account and a little knowledge of git:

1. [Fork](#) our repository
2. Git pull your fork

3. Make your changes, commit and push them back to GitHub
4. Create a [pull request](#) to inform us of your changes
5. After we've reviewed and accepted your work, we will merge your commits and the documentation will be updated automatically

All documentation is written in Sphinx *restructured text*. Behind the scenes we use ReadTheDocs.org to automatically publish the documentation. Below are some useful links to documentation on these techniques and systems:

- A simple Sphinx cheatsheet: [cheatsheet](#)
- [GitHub's git cheat sheet](#)

However, don't worry too much about all this new information. You'll see that Sphinx is quite intuitive. Besides, we will check your changes too. If you find it all too overwhelming anyway, or if you just don't feel like going through all this hustle, you are always free to just send us an e-mail with your remarks and we will change the documentation ourselves: helpdesk@surfsara.nl

8.6.2 Overview

The philosophy of Sphinx documentation is that content is stored in files that can be easily read *and* edited by humans, in a format called *restructured text*, with the file extension `.rst`. Using a simple grammar, text can be styled. The document is structured using special tags; using these tags, documentation can be split into multiple files, and you can cross-reference between files and build indexes.

8.6.3 Editing / preview

Because the syntax of the files is human readable, you can edit the files using your favorite text editor. Once you are done editing, you can generate documentation in various formats, such as HTML or epub. While you can edit the pages on virtually any system, it is recommended to preview your changes before publishing them.

There are different ways to generate the HTML documentation from source and review your changes:

- Docker image
- Sphinx local installation
- GitHub edit/preview

Note that you only need to use one of the options mentioned above. Using Docker is the preferred way, as this mimics the ReadTheDocs build system closest. GitHub edit/preview on the other hand is good enough for minor, textual changes, but is otherwise the least preferred option.

Docker image

This is the preferred option to build and test your changes. It tries to build the documentation the same way as readthedocs.org.

- Setup the Docker image. The instructions for different OS are [here](#)
- Once build, you can use it to build your documentation in the same build environment as used by their production server:

```
./build.sh
```

Note: For Mac OS X, please use `./build_mac.sh` instead.

Optionally you can provide an output location (default: `./build`) and the docker image name (default: `rtfd-build:base`):

```
./build.sh /alternative/output/path/ docker_image_alternative_name
```

Sphinx local installation

For the Sphinx documentation setup locally you will need to:

- Install Sphinx to your computer. The instructions for different OS are [here](#)
- `Clone` the current repo locally to fetch the source code or `pull` to update your local copy with the latest changes.
- To generate HTML documentation, use the command:

```
make html
```

which will generate static pages in the `build`-directory as long as you have the software Sphinx installed locally.

Github edit/preview

For small changes you can edit a page directly from the GitHub repo. The *preview* button does not give a fully compatible *rst* overview, but is sufficient for textual changes.

8.6.4 Publish changes

When you are done with your changes, commit and push to GitHub. See [Contributing through GitHub](#) on how to push your changes to our documentation.

8.7 Cookiebeleid

8.7.1 Wat is een cookie?

Wij maken op deze website gebruik van cookies. Een cookie is een eenvoudig klein bestandje dat met pagina's van deze website wordt meegestuurd en door uw browser op uw harde schijf van uw computer wordt opgeslagen. De daarin opgeslagen informatie kan bij een volgend bezoek weer naar onze servers teruggestuurd worden.

8.7.2 Google Analytics

Via onze website wordt een cookie geplaatst van het Amerikaanse bedrijf Google, als deel van de “Analytics”-dienst. Wij gebruiken deze dienst om bij te houden en rapportages te krijgen over hoe bezoekers de website gebruiken. Google kan deze informatie aan derden verschaffen indien Google hiertoe wettelijk wordt verplicht, of voor zover derden de informatie namens Google verwerken. Wij hebben hier geen invloed op. Wij hebben Google niet toegestaan de verkregen analytics informatie te gebruiken voor andere Google diensten. De informatie die Google verzamelt wordt zo veel mogelijk geanonimiseerd. Uw IP-adres wordt nadrukkelijk niet meegegeven. De informatie wordt overgebracht naar en door Google opgeslagen op servers in de Verenigde Staten. Google stelt zich te houden aan de Safe Harbor principles en is aangesloten bij het Safe Harbor-programma van het Amerikaanse Ministerie van Handel. Dit houdt in dat er sprake is van een passend beschermingsniveau voor de verwerking van eventuele persoonsgegevens.

8.7.3 In- en uitschakelen van cookies en verwijdering daarvan

Meer informatie omtrent het in- en uitschakelen en het verwijderen van cookies kan je vinden in de instructies en/of met behulp van de Help-functie van jouw browser.

8.7.4 Meer informatie over cookies?

Op de volgende websites kan je meer informatie over cookies vinden:

- Cookierecht.nl
- Consumentenbond: “Wat zijn cookies?”
- Consumentenbond: “Waarvoor dienen cookies?”
- Consumentenbond: “Cookies verwijderen”
- Consumentenbond: “Cookies uitschakelen”
- Your Online Choices: “A guide to online behavioural advertising”

Indices and tables

- `genindex`
- `modindex`
- `search`