A practical Introduction to UNICORE

Mathias Dalheimer and Dirk Petry
Fraunhofer Institut für Techno- und Wirtschaftsmathematik,
Kaiserslautern, Germany
{dalheimer|petryd}@itwm.fhg.de

21 March 2006
Version 1.0

Contents

1 Introduction **2**

2 Architecture Overview **3**
2.1 UNICORE Client 3
2.2 Gateway 6
2.3 Network Job Supervisor 7
2.4 Target System Interface 9
2.5 Drawbacks and missing features 9

3 The UNICORE Installation **10**
3.1 Assumptions 10
3.1.1 Network Layout 10
3.2 Certificates 10
3.3 Creating your own certificates 12
3.4 Downloading the necessary software 14
3.5 Installing the Gateway 14
3.5.1 Build the software 14
3.5.2 Set up certificates and add unicore user 15
3.5.3 Edit the file “gateway.properties” 15
3.5.4 Edit the file “connections” 16
3.5.5 Set up log file directory and file permissions 16
3.5.6 Add Gateway to your UNICORE site list 16
3.5.7 Test 17

3.6 Installing an NJS . . . 17
3.6.1 Build the software and set up directories . . . 17
3.6.2 Edit file “njs.properties” . . . 18
3.6.3 Edit file “njs.idb” . . . 19
3.6.4 Install the UUDB . . . 19
3.6.5 Create local worker user and add an external user to the UUDB . . . 20
3.6.6 Adjusting njs_admin . . . 20
3.7 Installing a TSI . . . 20
3.7.1 Build the software . . . 20
3.7.2 Edit file “/usr/local/unicore/tsi/conf/tsi.properties” . . . 21
3.8 Starting the system . . . 22
3.8.1 Testing the incarnation database . . . 22
3.9 Shutting down the system . . . 23
3.10 Adding a second Vsite . . . 23
3.10.1 Edit file “njs.properties” for the second Vsite . . . 23
3.10.2 Edit file “connections” for the Gateway . . . 24
3.10.3 Start-up . . . 24
3.11 Enabling trust between two Vsites . . . 24
3.12 Adding a software resource . . . 25
3.12.1 Example: POV-Ray . . . 25

1 Introduction

The UNICORE system [1] was developed in Germany in order to simplify the access to supercomputing resources. UNICORE is short for ”UNIform access to COmputing REsources”: The users can use a simple client to access computing resources, instead of obtaining a shell login, transfering the files to the target machine manually, and starting the job using the site-specific commands. The UNICORE software is available from Sourceforge under the BSD license [2].
With UNICORE, the provider of a resource can provide an abstract way of accessing its resources: Since the UNICORE system abstracts from different batch systems, it is easy for a site administrator to wrap different installations in a common interface. In the Fraunhofer Resource Grid, we run several sites using the UNICORE middleware.
Although there are some documents describing the UNICORE components in great detail, we have not found a guide that helps the administrator to get started. With this text, we hope to close this gap in the UNICORE documentation. In the first part, we will present the architecture and general information we think a UNICORE site admin should know. The second part is a step-by-step guide

through an example installation as we have it running in the Fraunhofer Resource Grid. Please note that you should not use this configuration as-is in a production environment: since external users are executing programs, you should use auditing techniques and harden your system. These security considerations are out of the scope of this document.
Although the second part contains detailed steps for the UNICORE installation, you should have a good understanding of Linux and public key infrastructures [3][1]. If you find any mistakes in this document, please contact the authors.

2 Architecture Overview

The UNICORE system consists of several components which interact over the network, see figure 1: A provider site is called *Usite*. The Usite in this example consists of two resources, called *Vsites*. The user uses the UNICORE client [4] to access the resources. All connections need to pass the *Gateway* [7], which authenticates the user and routes the connections to one of the *Network Job Supervisors* (NJS) [8]. The NJS uses the *Incarnation Database* (IDB) [10] to map jobs on the local system. When the user is authorized as described in the *UNICORE User Database*, the NJS forwards the job to the *Target System Interface* (TSI) [8]. The TSI in turn uses the local resource management system of the resource, usually a batch system, to submit the job. In the following sections, we will investigate the components further.

2.1 UNICORE Client

The *client* [4] is a GUI for users that provides access to the system. It is implemented in Java and therefore available on many platforms. A user can define jobs within the client, and use it to submit the job. During job execution, the job can be monitored and canceled. After the job terminated, the client allows the download of the results.
The client can be used to define jobs composed of several subtasks, see figure 2. A simple workflow engine allows the user to specify loops, conditions and sequences. Each job needs to be assigned to a resource. If data transfers are needed, they can be modeled explicitly or handled implicitly by the UNICORE system.
For certain applications like Fluent (see figure 3) or Nastran, application-specific plugins are available [2]. It is also possible to integrate custom plugins into the client. This way, the usage of applications can be simplified for the end-user. We will introduce a plugin for the Povray raytracer at the end of this guide.

[1] For a quick introduction, please refer to `http://en.wikipedia.org/wiki/X.509`

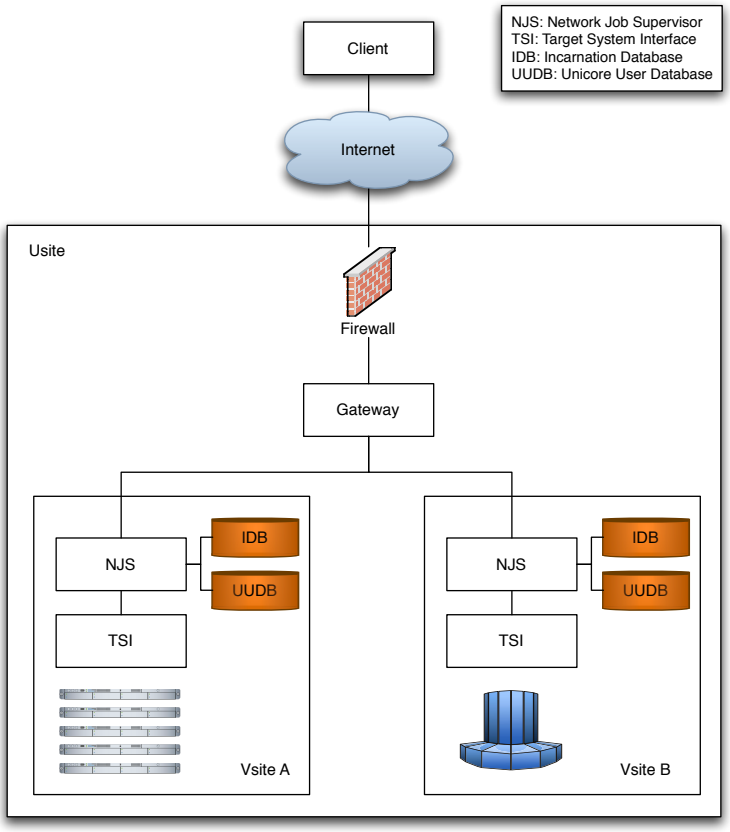

Figure 1: Example of a simple UNICORE infrastructure. A user can access two resources at the providers site.

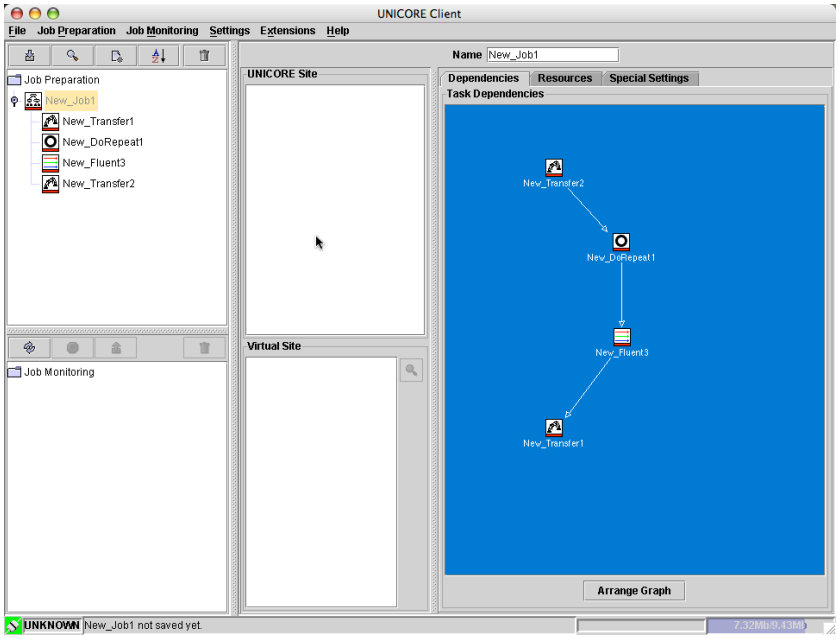

Figure 2: The workflow editor tab allows the modeling of complex jobs, composed of several actions. The user can embed subtasks in a workflow which will then be executed.

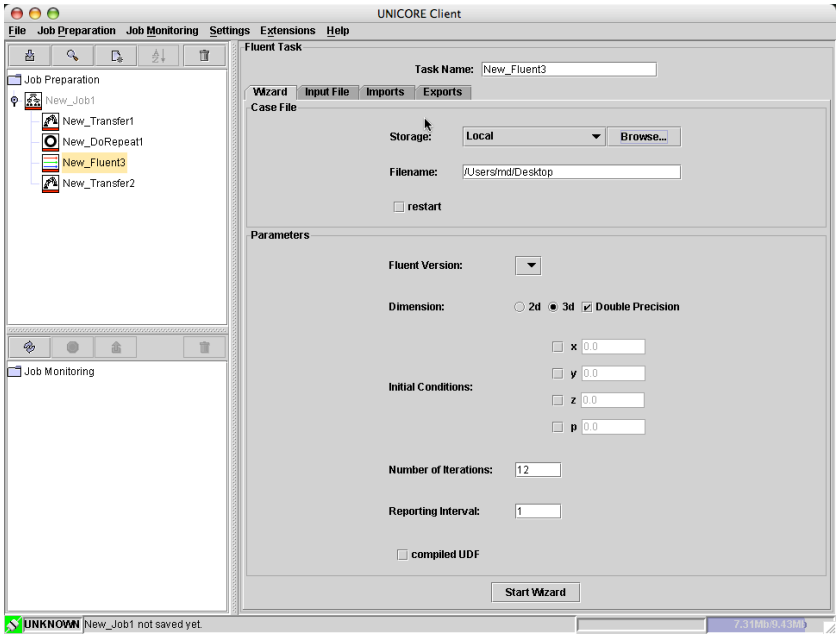

Figure 3: The client with the fluent plugin configuration page: the user doesn't need to specify how to execute fluent but instead focusses on fluent's parameters.

The client also handles the user authentication: the user only needs to import her X.509 certificates into the client keystore. During startup, the user needs to unlock the keystore, making her personal keys available to the client for the duration of the session. The client connections are encrypted using TLS, allowing the gateway to authenticate the connection using the user's X.509 certificates. From the user's viewpoint, the UNICORE client provides a single sign-on to the grid resources. In addition, all jobs are signed with the user's private key to prevent tampering.
During the execution of a job, there is no need for the user to stay online. Once a job is submitted, no interaction with the executing resources is needed since the complete workflow has been defined during job composition. As a drawback, it is also not possible to interfere with the job once it has been submitted - except for querying the state of subtasks and the cancelation of the whole job. Once the job has terminated, the client allows the download of the result files. These are stored on the resources until the user deletes them manually.
You can find more information about the client in the UNICORE client user manual [4]. If you want to try the client, we suggest you follow the instructions of the UNICORE Test Grid [5]. This site offers a free trial testbed that allows you to explore the possibilities of UNICORE without the need for a Usite setup.

2.2 Gateway

The *gateway* [7] provides access to a UNICORE site (Usite). A provider typically uses a single gateway to provide access to all his resources. The gateway receives incoming client connections and authenticates them. The administrator can select several trustworthy CAs, and users with certificates from one of these CAs will be authorized to connect to the gateway.
On the other side, the different Vsites connect to the gateway, waiting for user requests. It is possible to connect Vsites statically, but also dynamically. After the client has been authenticated, the gateway provides more information about the available systems to the client, i.e. the NJS that have registered with the gateway. When the client communicates with a NJS behind the gateway, the client sends the traffic to the gateway which in turn forwards it to the target NJS.
UNICORE uses the UNICORE Protocol Layer (UPL) for all network communication [6]. This communication includes the Abstract Job Object (AJO), which describes the DAG and the subtasks of the job. In addition, all file transfers are encapsulated. These data types are sent over an encrypted TLS connection, therefore all data is secured, see figure 4
The gateway reduces the network complexity significantly: there is only one port a client needs to connect to. The Vsites are independent of the gateway, allowing arbitrary network layouts behind the gateway. For a Usite to provide access to

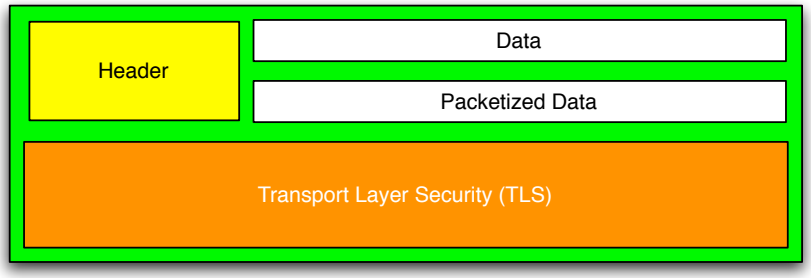

Figure 4: The UNICORE Protocol Layer (UPL) is based on a TLS-secured transport. The headers are used to signal the request type, while the data layer provides synchronous, the packetized data layer asynchronous messaging.

its resources over the internet, the firewall must allow connections to exactly one port.
You can find more information about the UNICORE gateway in the gateway documentation [7].

2.3 Network Job Supervisor

The *Network Job Supervisor* (NJS) [8] can be seen as an abstraction of a resource. It receives the Abstract Job Objects and concretizes them using the *Incarnation Database* (IDB) [10]. Using the *UNICORE User Database* (UUDB) [9], the NJS determines the user to use to start the job.
The UUDB stores the public keys of the users that have authorization to use the system. In addition, the UUDB allows the administrator to specify which local user will be used for the execution. This is the authorization database in UNICORE: The UUDB maps remote users to local users, called Xlogin. Therefore, the access rights of a remote user are the rights she has on the system as the user she is mapped to. A system administrator needs to ensure the local user has only the desired access rights on the system.
Before a user can submit a job to an NJS, the client ensures that the job requirements such as available memory, installed software etc. are satisfied. The definition of the NJS properties are made in the IDB: a simple text file that advertises the systems capabilities. The NJS provides the informations of the IDB to the client as part of the initial resource query. When a job's requirements are not satisfied, the job cannot be submitted and the subtask icon goes red (as you might have notices, this is the case in the figures 2 and 3). The job cannot be submitted to the NJS. If the requirements are satisfied, the icon goes green and the job is ready for submission.

Therefore, the IDB represents the capabilities of the NJS. Please note that no dynamic information like load, queue length or network bandwidth is stored in the IDB, but only static information.
When an AJO is submitted to the NJS, it is signed with the users private key. In order to verify the authenticity of the AJO, the NJS needs to look up the public key of the user in the UUDB. If the authenticity is given, the AJO needs to be concretized using the IDB: since the user cannot know where a binary, e.g. fluent, is installed, she only describes the job using abstract terms (e.g. FLUENT). In the IDB, those abstract terms are assigned to concrete values. This process is called *incarnation*.
The IDB provides also the possibility to include shell commands in these assignments. This provides a lot of flexibility for resource-specific command definitions. An administrator can test a software installation, and add a software advertisement to the IDB that is known to work. Please refer to section 3.12.1 (p. 25) for an example IDB entry. Of course, a user might also choose to stage in binaries or compile programs before execution.
An AJO is always submitted to one NJS. When the user specifies different execution locations for subtasks, the receiving NJS *consigns* the subtask to the respective Vsites. UNICORE differentiates two different X.509 signatures for AJOs [6]:

1. The *endorser* is the entity creating the job, i.e. the user. The endorsement determines how a job is executed at a Vsite.

2. The *consigner* is the entity submitting the job. This can be the user or an NJS forwarding the job to a different site. In the latter case, the consignment is necessary for the UPL layer connection verification: otherwise, the gateway of the receiving site would have no means to authenticate the connection.

Connected to job arriving at the NJS are input and output files which need to be stored. UNICORE distinguishes several file spaces [6]:

1. The *Uspace* is the job's working directory. Initially empty, the input files are stored here. During job execution, temporary and result files are added to the directory. When the result files are moved to the outcome file space, all remaining job files are deleted.

2. The *Spool* provides quasi-permanent storage for file transfers between jobs.

3. The *Outcome* is an area for result storage. The client can be used to retrieve these files.

4. The file system of the Vsite is available to jobs as well. For example, it is possible to use the Xlogin's home directory for file storage between jobs.

When an AJO is received, the input files are also sent over the UPL. The NJS dumps these files to the Uspace, using the TSI.
More information can be found in the NJS TSI documentation [8], the UUDB readme [9] and the IDB documentation [10].

2.4 Target System Interface

The *Target System Interface* (TSI) [8] takes concrete job requests and executes them on the target system, using the local user determined by the NJS. The TSI consists of a Perl script for maximum portability. It provides a simple abstraction of the target system. There are different TSI-Implementations available, e.g. to interface to the usual unix environment or to a cluster's batch system. The TSI connects to the NJS using a plain-text TCP connection.
When a job needs to be executed, the input files are received along with the Xlogin. The TSI then uses the local resource management system, e.g. Torque, to run the job. It can also provide the state of a running job and notifies the NJS upon job completion.
Beside various TSI implementations e.g. for Torque/Maui, Sun GridEngine and Platform LSF, there is also a Fork-TSI available: It will start jobs using a simple unix fork. This TSI can be used to connect small machines to the NJS.
You can find more information about the TSI in the NJS TSI documentation.

2.5 Drawbacks and missing features

Compared to other grid middlewares, there are two drawbacks:

1. There is no built-in resource broker: All subtasks need to be assigned to resources manually, as described in section 2.1. There is an external project called "UNICORE Resource Broker" [2], and other projects will develop a UNICORE broker.

2. Another lacking feature is sophisticated data handling: all files are staged in and out. But it should be possible to integrate e.g. the Storage Resource Broker (SRB) [11] into a job description, although we have no experience with this yet. There is also a package called "UniGrids ARFT" (Alternative Reliable File Transfer) on the UNICORE sourceforge page. This package seems to employ the Globus RFT protocol for file transfers in UNICORE.

3 The UNICORE Installation

In the example described here, we will install one Gateway, one NJS, and one TSI, i.e. one Usite and one Vsite. Adding additional Vsites follows exactly the same steps as the installation of the first.
Section 3.11 then describes the special case of enabling one NJS to consign jobs to a second NJS. And in section 3.12.1, we give an example of how to add a software resource to the IDB.

3.1 Assumptions

The example installation described in the following sections was made on a standard Debian Linux system. Adapting it to other Linux distributions should, however, be straight forward.
In particular, we chose to use the group "staff" for the `unicore` user which may not exist by default on other distributions. But you can easily add it using the groupadd command or replace it in the example by a different group of your choice. Otherwise, it is assumed that you have root access to the system where you would like to install the server.

3.1.1 Network Layout

As described in section 2, the UNICORE system consists of several daemons that interact over TCP connections. In this example installation, we use the network layout as show in figure 5. We will not cover the setup of the firewall and the other network components, but shortly describe the TCP connections UNICORE will establish in our setup. The client connects to the gateway on port 4004, which is the only port the firewall needs to allow access to. The gateway expects a NJS listening on port 4444 on usite and on port 4445 on testmachine1. The TSI-NJS communication needs two TCP connections: The TSI connects to the NJS on port 4666, the NJS to the TSI on port 4433. Please note that although these ports are the most frequently used ones, you can freely choose them during the installation.

3.2 Certificates

A major part of the installation work is to obtain the necessary certificates and keys to identify the elements of your UNICORE system. For details of the UNICORE security model refer to [12].
In the course of the installation, you will need at least three certificates/keys:

1. The certificate of the Certification Authority (CA) you would like to use: For a stand alone system, you can set up your own CA and generate this

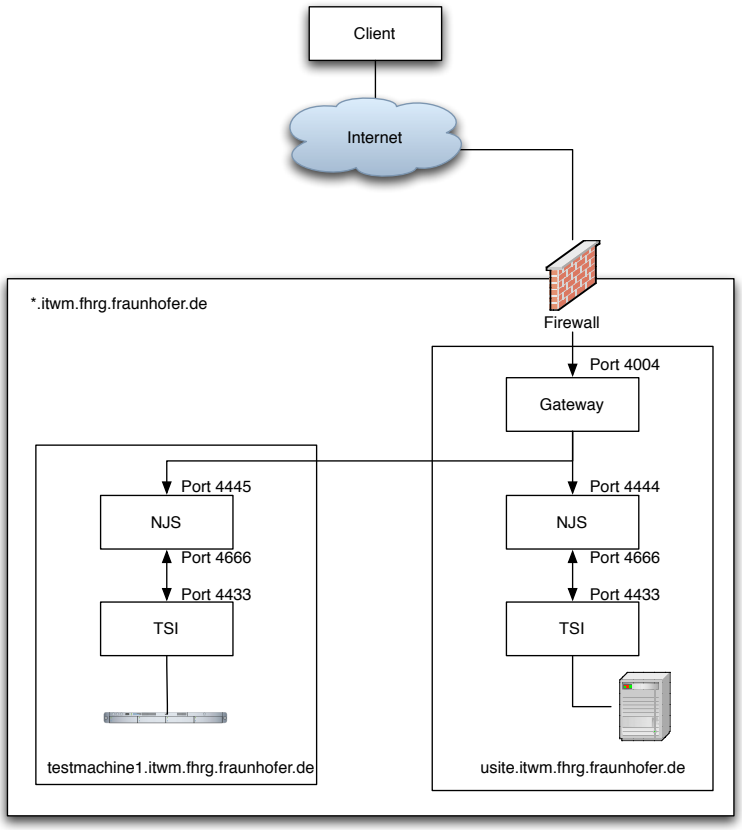

Figure 5: Network overview of the UNICORE installation at the Fraunhofer ITWM. Two machines are involved, usite and testmachine1.

certificate yourself using openssl. For a gateway that is to be part of an official UNICORE Grid, you will have to obtain the certificate from the CA accepted by that Grid.

In any case, you will need a file, which we call `mycacert.pem` in this example, containing the certificate of your CA.

2. The certificate of the host machine for your gateway signed by your CA: This certificate is needed in a so-called pkcs12 format keystore, a file which we will call `myusite.p12`.

3. The certificate for your Vsite, i.e. the NJS, signed by your CA: Since in this example, Gateway and NJS run on the same machine, you can re-use the file `myusite.p12`, but in the general case you need a separate certificate.

4. The certificate for at least one user who is going to connect to the Gateway via the Client, again signed by your CA: In this example we call this file `mypetrydcert.pem`. For the UNICORE Client, you will need in addition this certificate in a keystore, e.g. `mypetryd.p12`.

3.3 Creating your own certificates

Since obtaining the necessary certificates from an official CA can be cumbersome or at least time-consuming, we give here the necessary steps for generating the necessary certificates for a test setup. We assume that you have the standard openssl [13] installed. Please refer to the openssl manpages for more information, especially on the x509 commands.

1. Setting up your own certification authority and generate the CA certificate

   You should configure your openssl installation in `/etc/ssl/openssl.cnf` according to your needs. In an arbitrary directory, say `~/mykeys` on some machine, not necessarily the intended UNICORE server, as some arbitrary user (not root):

   ```
   $ > cd ~/mykeys
   $ > openssl req -config /etc/ssl/openssl.cnf -new -x509 \
      -keyout mycakey.pem -out mycacert.pem -days 365
   ```

   produces `mycakey.pem` and `mycacert.pem`. We will need `mycacert.pem` later.

```
$ > mkdir demoCA
$ > cd demoCA
$ > mkdir private
$ > mkdir newcerts
$ > touch index.txt
$ > cat > serial
01
ctrl-D
$ > ln -sf ../mycacert.pem cacert.pem
$ > cd private
$ > ln -sf ../../mycakey.pem cakey.pem
```

2. Generate the host machine certificate for your gateway

```
$ > cd ~/mykeys
$ > openssl req -nodes -config /etc/ssl/openssl.cnf -new \
   -keyout myusitekey.pem -out myusitereq.pem -days 356
$ > openssl ca -config /etc/ssl/openssl.cnf -policy policy_anything \
   -out myusitecert.pem -infiles myusitereq.pem
$ > openssl pkcs12 -export -in myusitecert.pem -inkey myusitekey.pem \
   -out myusite.p12
```

   produces `myusite.p12` and `myusitecert.pem` which we will need later.

3. Generate the certificate for a user

   (In this example, the user is petryd.)

```
$ > cd ~/mykeys
$ > openssl req -config /etc/ssl/openssl.cnf -new -keyout mypetrydkey.pem \
   -out mypetrydreq.pem -days 365
$ > openssl ca -config /etc/ssl/openssl.cnf -policy policy_anything \
   -out mypetrydcert.pem -infiles mypetrydreq.pem
$ > openssl pkcs12 -export -in mypetrydcert.pem -inkey mypetrydkey.pem \
   -out mypetryd.p12
```

   For later, we will need `mypetrydcert.pem`. The file `mypetryd.p12` will have to be imported into the keystore of your UNICORE Client (see [4]).

3.4 Downloading the necessary software

Using your favourite browser, go to

`http://www.unicore.org/downloads.htm`

and find the links to download the UNICORE elements via the sourceforge website:

1. The Gateway source, in this example:

   `http://mesh.dl.sourceforge.net/sourceforge/unicore/gateway_4.1.1_build_3_src.tar.gz`

2. The network job supervisor (NJS) source, in this example:

   `http://mesh.dl.sourceforge.net/sourceforge/unicore/njs_4.6.2_build_1_src.tar.gz`

3. The UNICORE user data base (UUDB) source

   `http://mesh.dl.sourceforge.net/sourceforge/unicore/uudb_1.0.0_src.tar.gz`

4. The target system interface (TSI) source, in this example

   `http://mesh.dl.sourceforge.net/sourceforge/unicore/tsi_4.1.2_build_1_src.tar.gz`

Furthermore, Java 1.4 and Ant (e.g. version 1.6) are required.
In the following we assume that all tar files were downloaded into an arbitrary directory named `scratch` .

3.5 Installing the Gateway

3.5.1 Build the software

As root:

```
# > cd scratch
# > tar xvzf  gateway_4.1.1_build_3_src.tar.gz
# > cd gateway_4.1.1_build_3_src
# > ant
# > mkdir /usr/local/unicore
# > mv build /usr/local/unicore/gateway
# > cd /usr/local/unicore
# > mkdir keystore
```

3.5.2 Set up certificates and add unicore user

Copy the following certificates to directory keystore

- Certificate of the Certification Authority
- Certificate of this Gateway signed by the above Certification Authority in pkcs12 format (here, e.g., usite.p12)

Add the user which is to run the Gateway and make him/her member both of the users and the staff group:

```
# > adduser --home /home/unicore --ingroup users unicore
# > gpasswd -a unicore staff
```

3.5.3 Edit the file "gateway.properties"

```
# > cd /usr/local/unicore/gateway/conf
# > ls -l
  total 12
  -rw-r--r--  1 root staff  394 Feb 10 16:16 connections
  -rw-r-----  1 root staff 2247 Feb 10 16:16 gateway.properties
  drwxr-x---  2 root staff 4096 Feb 10 16:16 logs
```

The following settings should be modified in `gateway.properties` :

`gw.gateway_host_name=usite.itwm.fhrg.fraunhofer.de`

(Note: in case of nameservice problems, a numerical IP address can be given.)

`gw.port = 4004`

`gw.connections = /usr/local/unicore/gateway/conf/connections`

`gw.identity=/usr/local/unicore/keystore/myusite.p12`

`gw.password=gateway2`

Note: `gw.password` is the export password necessary to open the keystore file given by `gw.identity` . You should make sure this file is not readable by everyone.

`gw.trusted_cas=/usr/local/unicore/keystore/mycacert.pem`

Note: in this example, two trusted CA certificates are given, needed is only one, the one that issued the certificate in `gw.identity` .

3.5.4 Edit the file "connections"

By default this file only contains comments. Add a line in the format

```
<Vsite name> <NJS machine> <NJS port>
```

e.g.

```
VSITE-LOCAL usite.itwm.fhrg.fraunhofer.de 4444
```

(Note: in case of nameservice problems, a numerical IP address can be given.) The Gateway alternatively offers a dynamic Vsite registration feature which is discussed in the documentation [7].

3.5.5 Set up log file directory and file permissions

As root:

```
# > cd /usr/local/unicore
# > chgrp -R staff gateway
# > cd gateway
# > chmod g+w conf
# > cd /usr/local/unicore/gateway/conf
# > rmdir logs
# > mkdir ~unicore/gateway-logs
# > ln -sf  ~unicore/gateway-logs logs
# > cd ~unicore
# > chown unicore.staff gateway-logs
# > chmod g+w gateway-logs
```

3.5.6 Add Gateway to your UNICORE site list

You have to tell your UNICORE Client which Usites to contact. On your Client machine, edit the XML file which contains your Usite list and add the lines:

```
<Usite name = "ITWM Usite" description = "The official Fraunhofer
       ITWM UNICORE gateway"
       address = "usite.itwm.fhrg.fraunhofer.de" port= "4004">
</Usite>
```

(Note: in case of nameservice problems, a numerical IP address can be given.)

3.5.7 Test

In order to test whether the installation is correct, start the Gateway **as user "unicore"** using exactly the following two lines:

```
$ > cd /usr/local/unicore/gateway/conf
$ > ../bin/start_gateway
```

You should see the single line response "Gateway started.". User "unicore" should see as a response to "ps x" that a job

```
java com.fujitsu.arcon.gateway.Gateway . gateway.prop
```

is running.
In the directory /usr/local/unicore/gateway/conf/logs you should see a new file named "GatewayLog..." and an empty file "startup.log" (if the default logging level was not changed).
At the end of the GatewayLog file, you should see the text "Initialisation complete.".
Stop the gate way using

```
$ > cd /usr/local/unicore/gateway/conf
$ > ../bin/stop_gateway
```

3.6 Installing an NJS

3.6.1 Build the software and set up directories

As root:

```
# > cd scratch
# > tar xvzf njs_4.6.2_build_1_src.tar.gz
# > cd njs_4.6.2_build_1_src
# > ant
# > mv build /usr/local/unicore/njs

# > cd  /usr/local/unicore
# > chgrp -R staff njs
# > cd njs
# > chmod g+w conf
```

```
# > cd conf
# > rmdir logs
# > mkdir ~unicore/njs-logs
# > ln -sf ~unicore/njs-logs/ logs
# > cd ~unicore
# > chown unicore.staff njs-logs
# > chmod g+w njs-logs
# > mkdir NJS_STATE
# > chown unicore.staff NJS_STATE
# > chmod g+w NJS_STATE
# > mkdir njs-space
# > chown unicore.users  njs-space
# > chmod g+w njs-space
```

3.6.2 Edit file "njs.properties"

Edit file `/usr/local/unicore/njs/conf/njs.properties` :

```
njs.vsite_name=VSITE-LOCAL

njs.save_dir=/home/unicore/NJS_STATE

uudb.directory=/usr/local/unicore/uudb

njs.incarnationdb=njs.idb

njs.admin_port=4555

njs.gateway_port=4444

njs.gateway=usite.itwm.fhrg.fraunhofer.de

njs.use_ssl=true

njs.ssl_password=gateway2
```

Again, you should make sure this file is not readable by anyone.

```
njs.njs_cert_loc=/usr/local/unicore/keystore/myusite.p12

njs.unicore_ca_loc=/usr/local/unicore/keystore/mycacert.pem
```

Note: we are using the same certificate for Gateway and NJS because both are running on the same machine. If your want to run the NJS on a different machine, you will need a separate certificate and set `njs.njs_cert_loc` accordingly.

3.6.3 Edit file "njs.idb"

```
# > cd /usr/local/unicore/njs/conf
# > cp example_linux.idb njs.idb
```

Edit file `njs.idb` :

```
-DEFINE NJS_FILE_SPACE /home/unicore/njs-space

TextInfoResource [Execution System]    Tag [Vsite] Value [Xen Debian]

TextInfoResource [Architecture]        Tag [Typ]   Value [Intel]

NAME USITE

SOURCE usite 4666 4433

QSTAT_XLOGIN unicore

-DEFINE TOUCH_CMD USR_DIR/touch

-DEFINE FIND_CMD  USR_DIR/find

-DEFINE PERL_CMD  /usr/bin/perl

INVOCATION [ ./<RUNCOMMAND> ]

-DEFINE TSI_LS /usr/local/unicore/tsi/tsi_NOBATCH/tsi_ls
```

3.6.4 Install the UUDB

As user root:

```
# > cd scratch
# > cd uudb_1.0.0_src
```

```
# > ant
# > cd build/src
# > chmod u+x installer
# > mkdir /usr/local/unicore/uudb
# > ./installer /usr/local/unicore/uudb /usr/local/unicore/njs
# > cd /usr/local/unicore
# > chmod g+rx uudb
# > cd /usr/local/unicore/uudb
# > chmod go+r UUDB
```

3.6.5 Create local worker user and add an external user to the UUDB

Copy the certificate of the external user into the directory `/usr/local/unicore/keystore` , e.g. `mypetrydcert.pem` . Edit the file and extract the part from `-----BEGIN CERTIFICATE-----` to `-----END CERTIFICATE-----` and save in a file, e.g. `petryd.pem` .
As user root:

```
# > adduser --home /home/igor --ingroup users igor
# > cd /usr/local/unicore/uudb
# > bin/add ../keystore/petryd.pem igor
```

Note: In the UNICORE Client, you have to set your identity for this Vsite to the same user as declared here.
A description of how to use the UUDB can be found in

```
uudb_1.0.0_src/build/src/README
```

3.6.6 Adjusting njs_admin

In the important script `/usr/local/unicore/njs/bin/njs_admin` , it is expected that perl is in `/bin/perl` . Should this not be the case, you have to edit the script and correct the path.
Furthermore, the values of `njs_port` and `njs_machine` have to be set at the top of the script:

```
$njs_port = "4555";
$njs_machine = "usite.itwm.fhrg.fraunhofer.de";
```

3.7 Installing a TSI

3.7.1 Build the software

As user root

```
# > cd scratch
# > tar xvzf  tsi_4.1.2_build_1_src.tar.gz
# > cd  tsi_4.1.2_build_1_src
# > ant
# > mv build /usr/local/unicore/tsi
```

In order to complete the installation, additional scripts have to be executed and various decisions to be taken. See `/usr/local/unicore/tsi/README` for more details.
An example is given below:
As user root:

```
# > cd /usr/local/unicore/tsi
# > chmod u+x Install*
# > ./Install.sh
```

The Install script asks for a TSI to install.

```
1 = tsi/NOBATCH
```

Installation directory:

```
tsi_NOBATCH
```

Confirm by typing y.

```
# > cd tsi_NOBATCH
# > cp tsi.LINUX tsi
# > cd ..
# > ./Install_permissions.sh tsi_NOBATCH
```

3.7.2 Edit file "/usr/local/unicore/tsi/conf/tsi.properties"

```
tsi.path=/usr/local/unicore/tsi/tsi_NOBATCH

tsi.njs_machine=usite.itwm.fhrg.fraunhofer.de

tsi.njs_port=4666

tsi.my_port=4433
```

3.8 Starting the system

The system can be started using the following script.

```
#!/bin/sh
# Start up a UNICORE gateway, NJS, and TSI
UNICORE_USER=unicore
UNICORE_HOME=/usr/local/unicore
UNICORE_STARTUP_LOG=${UNICORE_HOME}/startup.log
date > $UNICORE_STARTUP_LOG

cd ${UNICORE_HOME}/gateway/conf
su -c ../bin/start_gateway $UNICORE_USER >> $UNICORE_STARTUP_LOG 2>&1

cd ${UNICORE_HOME}/njs/conf
su -c ../bin/start_njs $UNICORE_USER >> $UNICORE_STARTUP_LOG 2>&1

cd ${UNICORE_HOME}/tsi/conf
../bin/start_tsi >> $UNICORE_STARTUP_LOG 2>&1

date >> $UNICORE_STARTUP_LOG
echo
echo UNICORE started.
```

If the gateway, NJS, and/or TSI run on different machines, the corresponding parts of the script have to be executed individually.
Note: This is a minimal script to accomplish the task. Additions are necessary to, e.g. integrate this script into a general Linux runlevel control etc.

3.8.1 Testing the incarnation database

Once the system is running, a first test of whether the file `njs/conf/njs.idb` was set up correctly, can be performed by typing (as root):

```
# > cd /usr/local/unicore/njs/conf
# > ../bin/njs_admin test_commands igor
```

The paths given for the various shell commands should be verified and if necessary corrected in `njs.idb` .

3.9 Shutting down the system

The system can be shut down using the following script:

```
#!/bin/sh
# Shut down all UNICORE elements running on this computer
UNICORE_HOME=/usr/local/unicore
UNICORE_SHUTDOWN_LOG=${UNICORE_HOME}/shutdown.log
date > $UNICORE_SHUTDOWN_LOG
cd ${UNICORE_HOME}/njs/conf
../bin/njs_admin tsi refresh >> $UNICORE_SHUTDOWN_LOG 2>&1
../bin/njs_admin tsi stop >> $UNICORE_SHUTDOWN_LOG 2>&1
../bin/njs_admin stop now >> $UNICORE_SHUTDOWN_LOG 2>&1
cd ${UNICORE_HOME}/gateway/conf
../bin/stop_gateway >> $UNICORE_SHUTDOWN_LOG 2>&1
date >> $UNICORE_SHUTDOWN_LOG
echo
echo UNICORE shut down.
```

3.10 Adding a second Vsite

Assuming, there are two independent machines running one UNICORE gateway each with associated NJS and TSI. Let's call these machines "usite" (as above) and "testmachine1".
We now want to disable the Gateway on testmachine1 and let the NJS of testmachine1 work with the Gateway on usite.

3.10.1 Edit file "njs.properties" for the second Vsite

On "testmachine1" (see above), edit `/usr/local/unicore/njs/conf`:

```
njs.vsite_name=VSITE-TESTMACHINE1
njs.gateway_port=4445
```

Note: This port must be different from the port used for the first Vsite.

```
njs.gateway=usite.itwm.fhrg.fraunhofer.de
```

All other settings should remain the same if the Vsite was running properly before with a different Gateway.

3.10.2 Edit file "connections" for the Gateway

On "usite", edit `/usr/local/unicore/gateway/conf/connections` and add the lines:

```
# Vsite running on testmachine1
VSITE-TESTMACHINE1 testmachine1.itwm.fhrg.fraunhofer.de 4445
```

(Note: in case of nameservice problems, a numerical IP address can be given.)
Make sure that the certificate on "testmachine1" which is given by the value of `njs.njs_cert_loc` in file njs.properties is signed by a certification authority whose certificate was registered with the Gateway on "usite".

3.10.3 Start-up

The UNICORE elements on both machines should be started as described in section 3.8. Whether the elements on "usite" are started first or second doesn't matter. Both sites can be shut down or restarted independently.

3.11 Enabling trust between two Vsites

If one NJS tries to consign a sub-job to a second NJS *without* having been registered in the second NJS's UUDB, the error message in the UNICORE Client looks similar to this:

```
Message: Remote NJS error on consign:
UPL Reply reports an error:
-1:com.fujitsu.arcon.njs.Authoriser$UnauthorisedException:
Consignor <EMAILADDRESS=none, CN=UNICORE VSite1, ...> is not known
to the UUDB
```

In order to enable the execution of different sub-jobs of a given job on different Vsites, the NJSs of the Vsites to which subjobs are consigned have to have the certificates of the NJSs that are consigning the jobs.
E.g., in order to permit the NJS on "usite" to consign a sub-job to the NJS on "testmachine1" (see section 3.10), the following actions have to be taken:
Copy the certificate for usite (e.g. myusitecert.pem) to testmachine1.
Edit the certificate and extract the part from `-----BEGIN CERTIFICATE-----` to `-----END CERTIFICATE-----` and save in a file, e.g. `usite.pem` . Move the file to /usr/local/unicore/keystore on "testmachine1".
Then as user root:

```
# > cd /usr/local/unicore/uudb
# > bin/add_njs ../keystore/usite.pem igor
```

Note: an Xlogin (here "igor") must be given, but it is never used by the NJS. It is, however, necessary for adminitrational purposes, e.g. when you want to delete an entry from the UUDB. It is permitted to use the same Xlogin for several users and NJSs.
No restart of any UNICORE element is necessary to make the change to the UUDB take effect.

3.12 Adding a software resource

If a plug-in for the resource exists on the Client side, you have to find out what the specific interface is, in particular what the official name of the resource is and how it is invoked.
Then you have to add a `SOFTWARE_RESOURCE` and an `INVOCATION` entry to the IDB. The former has to be made in the `EXECUTION_TSI` section of the IDB, the latter in the `RUN` section of the IDB.

3.12.1 Example: POV-Ray

The plugin for the image rendering software POV-Ray is a standard part of the UNICORE Client distribution. However, the necessary IDB entries are not well documented. They are ...
... in the `EXECUTION_TSI` section of `njs.idb`:

```
SOFTWARE_RESOURCE APPLICATION povray 3.5
```

... in the `RUN` section of of `njs.idb`:

```
INVOCATION povray-3.5 [ echo "Executing POV-Ray...";
                        cp /home/unicore/povray/povray.ini .;
                        env DISPLAY=localhost:0.0;
                        /usr/bin/povray -d $POVRAY_INPUTFILE
]
```

In addition, the directory `/home/unicore/povray` should be created and the file `povray.ini` copied there from the default directory of the POV-Ray distribution. The above invocation assumes that the executable is in `/usr/bin`.

Acknowledgement

The authors would like to thank Christian Peter for valuable suggestions and advice. The Competence Center High Performance Computing at the Fraunhofer ITWM (`http://www.itwm.fhg.de`) is supporting this work.

License

References

[1] A. Streit, D. Erwin, Th. Lippert, D. Mallmann, R. Menday, M. Rambadt, M. Riedel, M. Romberg, B. Schuller, and Ph. Wieder. UNICORE - From Project Results to Production Grids. In L. Grandinetti, editor, *Grid Computing and New Frontiers of High Performance Processing*. Elsevier, 2005. Available at `http://arxiv.org/abs/cs.DC/0502090`.

[2] Unicore Sourceforge Homepage. `http://unicore.sourceforge.net`.

[3] RFC 3280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. `http://www.ietf.org/rfc/rfc3280.txt`.

[4] Unicore Client User Guide. Available at `http://unicore.sourceforge.net/docs/client_manual.pdf`.

[5] Unicore Test Grid. `http://www.unicorepro.com/`.

[6] Sven van de Berghe. Unicore Architecture and Server Components. *Unicore Tutorial, GGF11*, 2004. Available at `http://unicore.sourceforge.net/docs/ggf11_tutorial_arch.pdf`.

[7] Sven van de Berghe. Using the Unicore Gateway v4.0.1. 2004. Available at `http://unicore.sourceforge.net/docs/gateway_manual.pdf`.

[8] Sven van de Berghe. Using the NJS and TSI (v4). 2004. Available at `http://unicore.sourceforge.net/docs/njs_tsi_manual.pdf`.

[9] UUDB Readme. 2004. Available at `http://unicore.sourceforge.net/manuals_readmes.html`.

[10] Sven van de Berghe. Using the Incarnation Database (v.4.1). 2004. Grid Forum Document GFD-I.18, available at `http://www.ggf.org/documents/GFD.18.pdf`.

[11] Storage Resource Broker Homepage. `http://www.sdsc.edu/srb/index.php/Main_Page`.

[12] T. Goss-Walter, R. Letz, Th. Kentemich, H.-Ch. Hoppe, and Ph. Wieder. An Analysis of the UNICORE Security Model. 2003. Available at `http://unicore.sourceforge.net/docs/gateway_manual.pdf`.

[13] OpenSSL Homepage. `http://www.openssl.org/`.