NORDUGRID

NORDUGRID-MANUAL-10

23/6/2009

Chelonia Administrator's manual

Zsombor Nagy[*]

Jon Nilsen[†]

Salman Zubair Toor [‡]

[*] zsombor@niif.hu

[†] j.k.nilsen@usit.uio.no

[‡] salman.toor@it.uu.se

Contents

1 Installing a centralized system

This section shows how to install Chelonia on a freshly installed debian lenny machine.

Get the latest ARC code from the subversion (`http://svn.nordugrid.org/`):

```
$ svn co http://svn.nordugrid.org/repos/nordugrid/arc1/trunk arc1
```

Check the README file for all the dependencies, and install them. Here 'aptitude' is used, but of course on other Linux distribution different package management software may be used, and the name of the packages may differ as well.

```
$ sudo aptitude install build-essential
$ sudo aptitude install autoconf
$ sudo aptitude install gettext
$ sudo aptitude install cvs
$ sudo aptitude install libtool
$ sudo aptitude install pkg-config
$ sudo aptitude install libglibmm-2.4-dev
$ sudo aptitude install python-dev
$ sudo aptitude install swig
$ sudo aptitude install libxml2-dev
$ sudo aptitude install libssl-dev
```

Now, run the `autogen` and `configure` scripts (target directories for the installation may be specified with the `--prefix` option):

```
$ cd arc1
$ ./autogen.sh
$ ./configure --disable-java --disable-a-rex-service --disable-isi-service \
    --disable-charon-service --disable-compiler-service \
    --disable-paul-service --disable-sched-service

Unit testing:       yes
Java binding:       no
Python binding:     yes (2.5)

Available third-party features:

RLS:                no
GridFTP:            no
LFC:                no
RSL:                no
SAML:               no
MYSQL CLIENT LIB:   no
gSOAP:              no

Included components:
A-Rex service:      no
ISI service:        no
CHARON service:     no
HOPI service:       yes
SCHED service:      no
STORAGE service:    yes
PAUL service:       no
SRM client (DMC):   no
GSI channel (MCC):  no
```

Start with compiling it:

```
$ make
```

And then install it (you need to be root if you want to install it to the default location):

```
$ sudo make install
```

It may be a good idea to run `ldconfig` to refresh the list of libraries:

```
$ ldconfig
```

Check if the python packages are installed correctly:

```
$ python
Python 2.5.2 (r252:60911, Jan  4 2009, 17:40:26)
[GCC 4.3.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import arc
>>> import storage
>>> import arcom
>>>
```

If there is no error on importing any of the packages, everything is OK. If the packages cannot be imported, try to set the PYTHONPATH environment variable to point to location of the packages.

It is important to syncronize the time of the machine - the storage system will make decisions based on differences between timestamps. When deploying multiple machines, different times will cause problems.

Host (and user) certificates are required to run the system. For testing purposes the NorduGrid InstantCA solution can be used. The InstantCA is capable of creating a demo-CA with short lifetime. The URL of the InstantCA service is `https://vls.grid.upjs.sk/CA/instantCA`. Now, create 3 user and 3 host certificates. (for this example, set the Organization Name to 'knowarc', the Common name of the CA to 'storage-test', the name of the users to 'penny', 'billy' and 'hammer' and the name of the hosts to 'upsilon', 'omicron' and 'theta'. A password for the CA and passwords for the user certificates are needed as well, these should be at least 4 characters long).

Now, download the generated certificates and untar the archive file to see its contents:

```
$ ls -lR
.:
total 28
drwxr-xr-x 2 zsombor zsombor 4096 2009-03-19 12:11 CA
-rw-r--r-- 1 zsombor zsombor  963 2009-03-19 12:11 ca.key
-rw-r--r-- 1 zsombor zsombor  786 2009-03-19 12:11 ca.pem
drwxr-xr-x 2 zsombor zsombor 4096 2009-03-19 12:11 hostCerts
-rw-r--r-- 1 zsombor zsombor 1044 2009-03-19 12:11 readme
-rw-r--r-- 1 zsombor zsombor    3 2009-03-19 12:11 serial.srl
drwxr-xr-x 2 zsombor zsombor 4096 2009-03-19 12:11 userCerts

./CA:
total 8
-rw-r--r-- 1 zsombor zsombor 786 2009-03-19 12:11 66fe707c.0
-rw-r--r-- 1 zsombor zsombor 139 2009-03-19 12:11 66fe707c.signing_policy

./hostCerts:
total 24
-rw-r--r-- 1 zsombor zsombor 786 2009-03-19 12:11 hostcert-omicron.pem
```

```
-rw-r--r-- 1 zsombor zsombor 782 2009-03-19 12:11 hostcert-theta.pem
-rw-r--r-- 1 zsombor zsombor 786 2009-03-19 12:11 hostcert-upsilon.pem
-rw-r--r-- 1 zsombor zsombor 887 2009-03-19 12:11 hostkey-omicron.pem
-rw-r--r-- 1 zsombor zsombor 891 2009-03-19 12:11 hostkey-theta.pem
-rw-r--r-- 1 zsombor zsombor 887 2009-03-19 12:11 hostkey-upsilon.pem

./userCerts:
total 24
-rw-r--r-- 1 zsombor zsombor 778 2009-03-19 12:11 usercert-billy.pem
-rw-r--r-- 1 zsombor zsombor 778 2009-03-19 12:11 usercert-hammer.pem
-rw-r--r-- 1 zsombor zsombor 778 2009-03-19 12:11 usercert-penny.pem
-rw-r--r-- 1 zsombor zsombor 963 2009-03-19 12:11 userkey-billy.pem
-rw-r--r-- 1 zsombor zsombor 963 2009-03-19 12:11 userkey-hammer.pem
-rw-r--r-- 1 zsombor zsombor 951 2009-03-19 12:11 userkey-penny.pem
```

You can see the certificate and key files for all the hosts and users, and the CA file with the proper hashed name. There is no rule about where to put these certificate, but it is common to put them in `/etc/grid-security`. Now, create this directory, and a subdirectory called `certificates`. To use the 'theta' host certificates for this machine, copy the corresponding certificate and key there:

```
$ sudo cp hostCerts/hostcert-theta.pem /etc/grid-security/hostcert.pem
$ sudo cp hostCerts/hostkey-theta.pem /etc/grid-security/hostkey.pem
$ sudo cp CA/* /etc/grid-security/certificates/
$ ls -lR /etc/grid-security/
/etc/grid-security/:
total 12
drwxr-xr-x 2 root root 4096 2009-03-19 12:36 certificates
-rw-r--r-- 1 root root  782 2009-03-19 12:35 hostcert.pem
-rw-r--r-- 1 root root  891 2009-03-19 12:36 hostkey.pem

/etc/grid-security/certificates:
total 8
-rw-r--r-- 1 root root 786 2009-03-19 12:36 66fe707c.0
-rw-r--r-- 1 root root 139 2009-03-19 12:36 66fe707c.signing_policy
```

Choose a user certificate and put it into the `~/.arc` directory, removing the password from the key file for convenience:

```
$ mkdir ~/.arc
$ cp userCerts/usercert-billy.pem ~/.arc/usercert.pem
$ openssl rsa -in userCerts/userkey-billy.pem -out ~/.arc/userkey.pem
$ chmod 600 ~/.arc/userkey.pem
$ ls -l ~/.arc
total 8
-rw-r--r-- 1 zsombor zsombor 778 2009-03-19 13:06 usercert.pem
-rw------- 1 zsombor zsombor 891 2009-03-19 13:09 userkey.pem
```

The storage system runs within the `arched` hosting environment daemon, which needs a configuration file describing which services to be run, on which ports do to listen, etc. Copy the template configuration files to the `/etc/arc` directory:

```
$ sudo mkdir /etc/arc
$ sudo cp arc1/src/services/storage/storage_service.xml.example /etc/arc/storage_service.xml
$ sudo cp arc1/src/services/storage/clientsslconfig.xml /etc/arc
$ ls -l /etc/arc
total 12
-rw-r--r-- 1 root root  259 2009-03-19 13:26 clientsslconfig.xml
-rw-r--r-- 1 root root 4606 2009-03-19 13:25 storage_service.xml
```

The template configuration file specifies several directories where the services can store their data. Create these directories at, e.g., /var/spool/arc.

```
$ sudo mkdir /var/spool/arc/ahash_data
$ sudo mkdir /var/spool/arc/shepherd_data
$ sudo mkdir /var/spool/arc/shepherd_store
$ sudo mkdir /var/spool/arc/shepherd_transfer
$ ls -l /var/spool/arc
total 16
drwxr-xr-x 2 root root 4096 2009-03-19 13:42 ahash_data
drwxr-xr-x 2 root root 4096 2009-03-19 13:42 shepherd_data
drwxr-xr-x 2 root root 4096 2009-03-19 13:42 shepherd_store
drwxr-xr-x 2 root root 4096 2009-03-19 13:42 shepherd_transfer
```

In this deployment the daemon will be run as root, so it is OK that the root owns these directories. If creating these directories somewhere else, then modify the paths in the storage_service.xml file.

If putting the host certificates and the CA certificate somewhere else than /etc/grid-security then modify the storage_service.xml (look for KeyPath, CertificatePath and CACertificatesDir). Also, modify /etc/arc/clientsslconfig.xml. This file contains the credentials which are used by the services when they connect to an other service. If putting clientsslconfig.xml somewhere else than /etc/arc, you need to modify all the ClientSSLConfig tags in the storage_service.xml.

Now, run the arched daemon with this config (specified by the -c option) first in the foreground (using the -f option) to see immediately if everything is right:

```
$ sudo /usr/local/sbin/arched -c /etc/arc/storage_service.xml -f
```

The template configuration file specifies a loglevel which only prints error messages, so nothing should be written on the screen now. The arched daemon should now listen on the ports: 60001 for the data transfer service called Hopi, and 60000 for all the other services:

```
$ netstat -at | grep LISTEN
tcp        0      0 *:60000                 *:*                     LISTEN
tcp        0      0 *:60001                 *:*                     LISTEN
```

Now it is time to set up the prototype client tool to access the system. It is currently called arc_storage_cli and it is by default installed to /usr/local/bin. The CLI tool needs to be told where it can find the user credentials, and a URL of a Bartender service. The Bartender service is the front-end of the storage system. This can be specified either by environment variables, or by a configuration file called ~/.arc/client.xml. Create the following client.xml file:

```
$ cat ~/.arc/client.xml
<ArcConfig>
  <KeyPath>/home/<username>/.arc/userkey.pem</KeyPath>
  <CertificatePath>/home/<username>/.arc/usercert.pem</CertificatePath>
  <CACertificatesDir>/etc/grid-security/certificates</CACertificatesDir>
  <BartenderURL>https://localhost:60000/Bartender</BartenderURL>
</ArcConfig>
```

Now you can use the arc_storage_cli to access your local deployment. Try to do a 'list' on the root collection ('/'):

```
$ arc_storage_cli
Usage:
  arc_storage_cli <method> [<arguments>]
Supported methods:
  stat - get detailed information about an entry
  makeCollection | make | mkdir - create a collection
  unmakeCollection | unmake | rmdir - remove an empty collection
  list | ls - list the content of a collection
  move | mv - move entries within the namespace
  putFile | put - upload a file
  getFile | get - download a file
  delFile | del | rm - remove a file
  modify | mod - modify metadata
  policy | pol - modify access policy rules
  unlink - remove a link to an entry from a collection without removing the entry itself
  credentialsDelegation | cre - delegate credentials for using gateway
  removeCredentials | rem - remove previously delegated credentials
Without arguments, each method prints its own help.

$ arc_storage_cli list
Usage: list <LN> [<LN> ...]

$ arc_storage_cli list /
- Calling the Bartender's list method...
[2009-03-19 18:37:12] [Arc.Loader] [ERROR] [12491/152991544]
    Component tcp.client(tcp) could not be created
[2009-03-19 18:37:12] [Arc.Loader] [ERROR] [12491/152991544]
    Component tls.client(tls) could not be created
[2009-03-19 18:37:12] [Arc.Loader] [ERROR] [12491/152991544]
    Component http.client(http) could not be created
[2009-03-19 18:37:12] [Arc.Loader] [ERROR] [12491/152991544]
    Component soap.client(soap) could not be created
ERROR: Wrong status from server.
Maybe wrong URL: 'https://localhost:60000/Bartender'
```

If you get the error 'Component [...] could not be created', try to specify to the python environment the location of the python packages:

```
$ export PYTHONPATH=/usr/local/lib/python2.5/site-packages/
```

Without arguments, the `arc_storage_cli` prints the list of available methods. Specifying only a method name prints the syntax of the given method. The `list` method requires Logical Names (LN). In the storage system, every file and collection has a Logical Name, which is a path within the namespace.

Now a (centralized) storage service is running, including a Hopi service, which is a basic HTTP server, for transferring files. The Bartender service is the front-end of the system to the users, so the clients always connect to the Bartender service. There are several directories configured in the config file where the services store their data. These directories can always be emptied, which resets the whole system to a clean state. If Chelonia is running with clean data directories, then it is completely empty, meaning that there is not even a root collection:

```
$ arc_storage_cli list /
- Calling the Bartender's list method...
- done in 0.34 seconds.
'/': not found
```

The root collection has the Logical Name of '/', and it's role to provide a starting point for all the other Logical Names. This root collection needs to be created first:

```
$ arc_storage_cli makeCollection /
- Calling the Bartender's makeCollection method...
- done in 0.81 seconds.
Creating collection '/': done
```

```
$ arc_storage_cli list /
- Calling the Bartender's list method...
- done in 0.55 seconds.
'/': collection
    empty.
```

Now there is an empty root collection.

2 Using the prototype client tool

Assume that Chelonia is running and you want to access it with the `arc_storage_cli` client. The `~/.arc/client.xml` is set properly, the contents of the root collection can be listed:

```
$ arc_storage_cli list /
- Calling the Bartender's list method...
- done in 0.55 seconds.
'/': collection
    empty.
```

Now, create a small file and upload it:

```
$ cat > orange
orange
$ cat orange
orange

$ arc_storage_cli put orange /
- The size of the file is 7 bytes
- Calculating md5 checksum...
- done in 0.0003 seconds, md5 checksum is 6aefd2842be62cd470709b27aedc7db7
- Calling the Bartender's putFile method...
- done in 1.26 seconds.
- Got transfer URL: http://localhost:60001/hopi/7fcef91e-ac9c-658d-4def-17d52ba33cfb
- Uploading from 'orange'
    to 'http://localhost:60001/hopi/7fcef91e-ac9c-658d-4def-17d52ba33cfb' with http...
    0 s:        0.0 kB       0.0 kB/s       0.0 kB/s    . . .
- done in 0.0182 seconds.
'orange' (7 bytes) uploaded as '/orange'.

$ arc_storage_cli list /
- Calling the Bartender's list method...
- done in 0.51 seconds.
'/': collection
  orange              <file>
```

Additional information about the file can be found with the stat method:

```
$ arc_storage_cli stat /orange
- Calling the Bartender's stat method...
- done in 0.40 seconds.
'/orange': found
  states
    checksumType: md5
    neededReplicas: 1
    size: 7
    checksum: 6aefd2842be62cd470709b27aedc7db7
  entry
    owner: /DC=eu/DC=KnowARC/O=knowarc/CN=billy
    GUID: 12ce831b-eab8-1f47-29f6-19fb85ebf46a
    type: file
  parents
    0/orange: parent
  locations
    https://localhost:60000/Shepherd 3e9ea612-87c0-1e81-58f3-cec6b0b125c1: alive
  timestamps
    created: 1237504821.91
```

Here, the owner of the file (Billy), the location of the replicas (currently only one), the number of needed replicas (which is also only one), the checksum and size of the file, and the timestamp of the creation is shown.

This file can be downloaded:

```
$ arc_storage_cli get /orange /tmp
- Calling the Bartender’s getFile method...
- done in 0.51 seconds.
- Got transfer URL: http://localhost:60001/hopi/521c5869-4c93-8e28-aa3a-03f477f34f63
- Downloading from ’http://localhost:60001/hopi/521c5869-4c93-8e28-aa3a-03f477f34f63’
    to ’/tmp/orange’ with http...
    0 s:        0.0 kB       0.0 kB/s       0.0 kB/s    . . .
- done in 1.0071 seconds.
’/orange’ (7 bytes) downloaded as ’/tmp/orange’.
$ cat /tmp/orange
orange
```

You can create collections which can contain files and other collections forming a tree-hierarchy. You can move files and collection around within this namespace.

```
$ arc_storage_cli makeCollection /fruits
- Calling the Bartender’s makeCollection method...
- done in 0.79 seconds.
Creating collection ’/fruits’: done

$ arc_storage_cli makeCollecion /fruits/apple
- Calling the Bartender’s makeCollection method...
- done in 0.91 seconds.
Creating collection ’/fruits/apple’: done

$ arc_storage_cli move /orange /fruits/
- Calling the Bartender’s move method...
- done in 0.99 seconds.
Moving ’/orange’ to ’/fruits/’: moved

$ arc_storage_cli list /fruits
- Calling the Bartender’s list method...
- done in 0.60 seconds.
’/fruits’: collection
  orange              <file>
  apple               <collection>
```

You can remove files, and remove collections (but only if the collection is empty):

```
$ arc_storage_cli rmdir /fruits
- Calling the Bartender's unmakeCollection method...
- done in 0.41 seconds.
Removing collection '/fruits': collection is not empty

$ arc_storage_cli rmdir /fruits/apple
- Calling the Bartender's unmakeCollection method...
- done in 0.89 seconds.
Removing collection '/fruits/apple': removed

$ arc_storage_cli rm /fruits/orange
- Calling the Bartender's delFile method...
- done in 0.89 seconds.
/fruits/orange: deleted

$ arc_storage_cli rmdir /fruits
- Calling the Bartender's unmakeCollection method...
- done in 0.79 seconds.
Removing collection '/fruits': removed
```

Now, introduce a new user, Penny, into the system. Again, `arc_storage_cli` can be told to use user certificate and key files for this user with modifying the `client.xml` or by specifying the `ARC_CERT_FILE` and `ARC_KEY_FILE` environment variables.

```
$ ARC_CERT_FILE=usercert-penny.pem ARC_KEY_FILE=userkey-penny.pem \
arc_storage_cli list /
- Calling the Bartender's list method...
- done in 0.31 seconds.
'/': denied

$ ARC_CERT_FILE=usercert-penny.pem ARC_KEY_FILE=userkey-penny.pem \
arc_storage_cli put /tmp/orange /pennys-orange
- The size of the file is 7 bytes
- Calculating md5 checksum...
- done in 0.0001 seconds, md5 checksum is 6aefd2842be62cd470709b27aedc7db7
- Calling the Bartender's putFile method...
- done in 0.81 seconds.
/pennys-orange: failed to add child to parent
```

Listing the root collection or uploading files into it for Penny is denied, because the root collection is owned by Billy, and nobody else has permissions to it. Now, Billy allows for everyone to access the root collection:

```
$ arc_storage_cli pol / change ALL +read +addEntry
- Calling the Bartender's stat method...
- done in 0.30 seconds.
old action list for the user ALL was ''
the new action list of user ALL for LN / will be '+read +addEntry'
- Calling the Bartender's modify method...
- done in 0.50 seconds.
set.
```

The syntax of the `policy` method is the following:

```
$ arc_storage_cli pol
Usage: policy <LN> <changeType> <identity> <action list>
  <changeType> could be 'set', 'change' or 'clear'
```

```
    'set': sets the action list to the given user overwriting the old one
    'change': modify the current action list with adding and removing actions
    'clear': clear the action list of the given user
  <identity> could be a '<UserDN>' or a 'VOMS:<VO name>'
  <action list> is a list actions prefixed with '+' or '-'
    e.g. '+read +addEntry -delete'
    possible actions:
        read addEntry removeEntry delete
        modifyPolicy modifyStates modifyMetadata
```

So Billy has changed the action list for user 'ALL' to '+read +addEntry', which means that everybody can list the contents of this collection, create subcollections in it, upload files into it, or move existing entries into it.

Now, Penny can upload a file:

```
$ ARC_CERT_FILE=usercert-penny.pem ARC_KEY_FILE=userkey-penny.pem \
arc_storage_cli put /tmp/orange /pennys-orange
- The size of the file is 7 bytes
- Calculating md5 checksum...
- done in 0.0001 seconds, md5 checksum is 6aefd2842be62cd470709b27aedc7db7
- Calling the Bartender's putFile method...
- done in 1.18 seconds.
- Got transfer URL: http://localhost:60001/hopi/4145891d-0e4c-b6bd-e4d6-dc80554b9d35
- Uploading from '/tmp/orange'
    to 'http://localhost:60001/hopi/4145891d-0e4c-b6bd-e4d6-dc80554b9d35' with http...
    1 s:        0.0 kB       0.0 kB/s       0.0 kB/s    . . .
- done in 1.0071 seconds.
'/tmp/orange' (7 bytes) uploaded as '/pennys-orange'.
```

This file now only can be downloaded by Penny, so she should allow for Billy to download it as well:

```
$ ARC_CERT_FILE=usercert-penny.pem ARC_KEY_FILE=userkey-penny.pem \
arc_storage_cli pol /pennys-orange change /DC=eu/DC=KnowARC/O=knowarc/CN=billy +read
- Calling the Bartender's stat method...
- done in 0.41 seconds.
old action list for the user /DC=eu/DC=KnowARC/O=knowarc/CN=billy was ''
the new action list of user /DC=eu/DC=KnowARC/O=knowarc/CN=billy for LN /pennys-orange
    will be '+read'
- Calling the Bartender's modify method...
- done in 0.59 seconds.
set.

$ arc_storage_cli stat /pennys-orange
- Calling the Bartender's stat method...
- done in 0.43 seconds.
'/pennys-orange': found
  locations
    https://localhost:60000/Shepherd 1cc3b115-2350-8334-d206-a7b891912c00: alive
  states
    checksumType: md5
    neededReplicas: 1
    size: 7
    checksum: 6aefd2842be62cd470709b27aedc7db7
  parents
    0/pennys-orange: parent
  timestamps
    created: 1237509633.17
  policy
    /DC=eu/DC=KnowARC/O=knowarc/CN=billy: +read
  entry
    owner: /DC=eu/DC=KnowARC/O=knowarc/CN=penny
    GUID: 0e6a4c9a-d4e7-e0dc-e083-16b83d460d82
    type: file
```

The metadata of the file shows that Billy has ‘read’ rights. Billy can now download this file:

```
$ arc_storage_cli get /pennys-orange /tmp
- Calling the Bartender’s getFile method...
- done in 0.50 seconds.
- Got transfer URL: http://localhost:60001/hopi/c16d7e12-42b0-7882-c09e-1cac31dbdbcd
- Downloading from ’http://localhost:60001/hopi/c16d7e12-42b0-7882-c09e-1cac31dbdbcd’
    to ’/tmp/pennys-orange’ with http...
    0 s:        0.0 kB       0.0 kB/s       0.0 kB/s    . . .
- done in 1.0065 seconds.
’/pennys-orange’ (7 bytes) downloaded as ’/tmp/pennys-orange’.
$ cat /tmp/pennys-orange
orange
```

The needed number of replicas for a new file can be specified with the ARC_NEEDED_REPLICAS environment variable.

```
$ ARC_NEEDED_REPLICAS=2 arc_storage_cli put orange /new-orange
- The size of the file is 7 bytes
- Calculating md5 checksum...
- done in 0.0001 seconds, md5 checksum is 6aefd2842be62cd470709b27aedc7db7
- Calling the Bartender’s putFile method...
- done in 1.11 seconds.
- Got transfer URL: http://localhost:60001/hopi/0ad54182-7618-a2af-c490-dea7c4d6fc31
- Uploading from ’orange’ to
    ’http://localhost:60001/hopi/0ad54182-7618-a2af-c490-dea7c4d6fc31’ with http...
    1 s:        0.0 kB       0.0 kB/s       0.0 kB/s    . . .
- done in 1.0057 seconds.
’orange’ (7 bytes) uploaded as ’/new-orange’.

$ arc_storage_cli stat /new-orange
- Calling the Bartender’s stat method...
- done in 0.40 seconds.
’/new-orange’: found
  states
    checksumType: md5
    neededReplicas: 2
    size: 7
    checksum: 6aefd2842be62cd470709b27aedc7db7
  entry
    owner: /DC=eu/DC=KnowARC/O=knowarc/CN=billy
    GUID: 4e673e28-4769-50af-f02e-d7a67632f87a
    type: file
  parents
    0/new-orange: parent
  locations
    https://localhost:60000/Shepherd 4aacd276-653c-3ca5-c1a2-78a876c129c9: alive
  timestamps
    created: 1237509989.43
```

Here, the ‘neededReplica’ metadata says 2, but the system is deployed in a centralized way for testing purposes, so there is only one storage element (which is a Shepherd service and a Hopi service together), so the file can only have one replica and there can only be one location.

The number of needed replicas of an existing file can be modified with the ‘modify’ method:

```
$ arc_storage_cli modify /new-orange set states neededReplicas 1
- Calling the Bartender’s modify method...
- done in 0.63 seconds.
set

$ arc_storage_cli stat /new-orange | grep neededReplicas
    neededReplicas: 1
```

It should be noted that the current version of the `arc_storage_cli` tool does not support recursive methods or working with multiple files at once.

3 Adding more storage elements

The following shows how to add a new storage element to the system. A new storage element should be on a different physical machine, preferably in a different building to minimize the possibility of both storage elements being offline at the same time. But for testing purposes it is of course possible to run it on the same machine using different ports. A storage element consists of a Shepherd service and a storage element service, which this time will be a ‘byteio’ service. The ‘byteio’ service implements a small part of the ByteIO interface, which means that it just put the entire file data into the SOAP message, so it is only suitable for small files.

Now, modify the configuration file of the original server. Currently all the transfer URLs which were generated when uploading and downloading files contained the hostname ‘localhost’. This is not OK if access from an other machine is needed. In this case, transfer URL’s must contain the external hostname of the machine. To do this, change this line in `/etc/arc/storage_service.xml`:

```
    <TURLPrefix>http://localhost:60001/hopi/</TURLPrefix>
```

to the real hostname of the machine. But this is not enough. Currently the Shepherd services uses their URLs as IDs, and the other services use these IDs to connect to a Shepherd service, so the ID of the Shepherd service needs to be changed to a real external URL as well. For the sake of clarity, change all ‘localhost’ entries in the config file to the real hostname. In this example, the name of this machine is just ‘storage’ so the Shepherd section should look like this:

```
<Service name="pythonservice" id="shepherd">
    <ClassName>storage.shepherd.shepherd.ShepherdService</ClassName>
    <ServiceID>https://storage:60000/Shepherd</ServiceID>
    <CheckPeriod>20</CheckPeriod>
    <MinCheckInterval>0.1</MinCheckInterval>
    <CreatingTimeout>600</CreatingTimeout>
    <StoreClass>arcom.store.cachedpicklestore.CachedPickleStore</StoreClass>
    <StoreCfg><DataDir>/var/spool/arc/shepherd_data</DataDir></StoreCfg>
    <BackendClass>storage.shepherd.hardlinkingbackend.HopiBackend</BackendClass>
    <BackendCfg>
        <DataDir>/var/spool/arc/shepherd_store</DataDir>
        <TransferDir>/var/spool/arc/shepherd_transfer</TransferDir>
        <TURLPrefix>http://storage:60001/hopi/</TURLPrefix>
    </BackendCfg>
    <LibrarianURL>https://storage:60000/Librarian</LibrarianURL>
    <BartenderURL>https://storage:60000/Bartender</BartenderURL>
    <ClientSSLConfig FromFile="/etc/arc/clientsslconfig.xml" />
</Service>
```

Now, check if the centralized deployment is still working, stop the arched daemon, remove the data directories to make it a clean start, and restart it with the modified config.

```
$ sudo rm -r /var/spool/arc/*
$ sudo /usr/local/sbin/arched -c /etc/arc/storage_service.xml -f
```

See if the client is still usable:

```
$ arc_storage_cli list /
- Calling the Bartender's list method...
- done in 0.31 seconds.
'/': not found

$ arc_storage_cli makeCollection /
- Calling the Bartender's makeCollection method...
- done in 0.63 seconds.
Creating collection '/': done

$ arc_storage_cli list /
- Calling the Bartender's list method...
- done in 0.50 seconds.
'/': collection
    empty.
```

Now, upload a file and ask for two replicas:

```
$ ARC_NEEDED_REPLICAS=2 arc_storage_cli put orange /
- The size of the file is 7 bytes
- Calculating md5 checksum...
- done in 0.0005 seconds, md5 checksum is 6aefd2842be62cd470709b27aedc7db7
- Calling the Bartender's putFile method...
- done in 1.10 seconds.
- Got transfer URL: http://storage:60001/hopi/139dc57d-8c12-c188-a4a6-973449553811
- Uploading from 'orange'
    to 'http://storage:60001/hopi/139dc57d-8c12-c188-a4a6-973449553811' with http...
    1 s:        0.0 kB       0.0 kB/s       0.0 kB/s    . . .
- done in 1.0090 seconds.
'orange' (7 bytes) uploaded as '/orange'.

$ arc_storage_cli stat /orange
- Calling the Bartender's stat method...
- done in 0.41 seconds.
'/orange': found
  states
    checksumType: md5
    neededReplicas: 2
    size: 7
    checksum: 6aefd2842be62cd470709b27aedc7db7
  entry
    owner: /DC=eu/DC=KnowARC/O=knowarc/CN=billy
    GUID: 3fdad88a-944f-79c5-4f23-c5b3d3bc981e
    type: file
  parents
    0/orange: parent
  locations
    https://storage:60000/Shepherd 0b46f53c-419d-ce4e-181a-ec95d16dcc95: alive
  timestamps
    created: 1237519441.96
```

Currently it has only one replica, because there is still only one storage element. Now, create a new config for the other machine, here called storage2:

```
<?xml version="1.0"?>
<ArcConfig
  xmlns="http://www.nordugrid.org/schemas/ArcConfig/2007"
  xmlns:tcp="http://www.nordugrid.org/schemas/ArcMCCTCP/2007"
>
    <Server>
        <Pidfile>/var/run/arched2.pid</Pidfile>
        <Logger level="ERROR">/var/log/arched2.log</Logger>
    </Server>
    <ModuleManager>
        <Path>/usr/local/lib/arc/</Path>
    </ModuleManager>
    <Plugins><Name>mcctcp</Name></Plugins>
    <Plugins><Name>mcctls</Name></Plugins>
    <Plugins><Name>mcchttp</Name></Plugins>
    <Plugins><Name>mccsoap</Name></Plugins>
    <Chain>
        <Component name="tcp.service" id="tcp">
            <next id="tls"/>
            <tcp:Listen>
                <tcp:Port>60006</tcp:Port>
                <tcp:Version>4</tcp:Version>
            </tcp:Listen>
        </Component>
        <Component name="tls.service" id="tls">
            <next id="http"/>
            <KeyPath>/etc/grid-security/hostkey.pem</KeyPath>
            <CertificatePath>/etc/grid-security/hostcert.pem</CertificatePath>
            <CACertificatesDir>/etc/grid-security/certificates</CACertificatesDir>
        </Component>
        <Component name="http.service" id="http">
            <next id="soap">POST</next>
        </Component>
        <Component name="soap.service" id="soap">
            <next id="plexer"/>
        </Component>
        <Plexer name="plexer.service" id="plexer">
            <next id="shepherd">^/Shepherd$</next>
            <next id="byteio">^/byteio/</next>
        </Plexer>
        <Service name="pythonservice" id="shepherd">
            <ClassName>storage.shepherd.shepherd.ShepherdService</ClassName>
            <ServiceID>https://storage2:60006/Shepherd</ServiceID>
            <CheckPeriod>20</CheckPeriod>
            <MinCheckInterval>0.1</MinCheckInterval>
            <CreatingTimeout>600</CreatingTimeout>
            <StoreClass>arcom.store.cachedpicklestore.CachedPickleStore</StoreClass>
            <StoreCfg>
                <DataDir>/var/spool/arc/shepherd_data2</DataDir>
            </StoreCfg>
            <BackendClass>storage.shepherd.byteio.ByteIOBackend</BackendClass>
            <BackendCfg>
                <DataDir>/var/spool/arc/shepherd_store2</DataDir>
                <TransferDir>/var/spool/arc/shepherd_transfer2</TransferDir>
                <TURLPrefix>https://storage2:60006/byteio/</TURLPrefix>
            </BackendCfg>
            <LibrarianURL>https://storage:60000/Librarian</LibrarianURL>
            <BartenderURL>https://storage:60000/Bartender</BartenderURL>
            <ClientSSLConfig FromFile="/etc/arc/clientsslconfig.xml"/>
        </Service>
        <Service name="pythonservice" id="byteio">
            <ClassName>storage.shepherd.byteio.ByteIOService</ClassName>
            <TransferDir>/var/spool/arc/shepherd_transfer2</TransferDir>
            <NotifyURL>https://storage2:60006/Shepherd</NotifyURL>
            <ClientSSLConfig FromFile="/etc/arc/clientsslconfig.xml"/>
        </Service>
    </Chain>
</ArcConfig>
```

This configuration contains only two services: a Shepherd, and byteio service. The byteio service is just a very simple storage element service which moves files within SOAP messages. The Shepherd service needs to have a backend for all supported storage element services, so it has a ByteIOBackend configured. This Shepherd connects to the first server's Bartender and Librarian services. (See the technical documentation for details about the services.)

While the original (`storage`) server still runs the services, start the new server:

```
$ sudo /usr/local/sbin/arched -c /etc/arc/storage_service_2.xml -f
```

After a minute or two, check the file again:

```
$ arc_storage_cli stat /orange
- Calling the Bartender's stat method...
- done in 0.41 seconds.
'/orange': found
  states
    checksumType: md5
    neededReplicas: 2
    size: 7
    checksum: 6aefd2842be62cd470709b27aedc7db7
  entry
    owner: /DC=eu/DC=KnowARC/O=knowarc/CN=billy
    GUID: 24b442eb-ab1a-680e-1f79-4c20479d3ea9
    type: file
  parents
    0/orange: parent
  locations
    https://storage:60000/Shepherd 315d59d3-50cd-1a10-02e6-8e6a528473c3: alive
    https://storage2:60006/Shepherd c1334c09-5d3d-983b-2081-87fe60d4764a: alive
  timestamps
    created: 1237519828.4
```

It indeed has two replicas now, one on each server.

Now, try to download it multiple times:

```
$ arc_storage_cli get /orange /tmp/orange1
- Calling the Bartender's getFile method...
- done in 0.50 seconds.
- Got transfer URL: http://storage:60001/hopi/de0d951f-7379-1ce4-0813-48a384c11982
- Downloading from 'http://storage:60001/hopi/de0d951f-7379-1ce4-0813-48a384c11982'
    to '/tmp/orange1' with http...
    0 s:        0.0 kB       0.0 kB/s       0.0 kB/s    . . .
- done in 1.0060 seconds.
```

```
’/orange’ (7 bytes) downloaded as ’/tmp/orange1’.

$ arc_storage_cli get /orange /tmp/orange2
- Calling the Bartender’s getFile method...
- done in 0.51 seconds.
- Got transfer URL: https://storage2:60006/byteio/52b6bffa-4812-03a5-93a1-834e38077ac5
- Downloading from ’https://storage2:60006/byteio/52b6bffa-4812-03a5-93a1-834e38077ac5’
    to ’/tmp/orange2’ with byteio...
- done in 0.2822 seconds.
’/orange’ (7 bytes) downloaded as ’/tmp/orange2’.
```

First it was downloaded from the `storage` server via HTTP, second time it was downloaded from the `storage2` server via ByteIO.

Now, create a third storage element on a machine called `storage3` this time using a Hopi service as storage element service like on the main `storage` server:

```
zsombor@storage:~$ cat /etc/arc/storage_service_3.xml
<?xml version="1.0"?>
<ArcConfig
  xmlns="http://www.nordugrid.org/schemas/ArcConfig/2007"
  xmlns:tcp="http://www.nordugrid.org/schemas/ArcMCCTCP/2007"
>
    <Server>
        <Pidfile>/var/run/arched3.pid</Pidfile>
        <Logger level="ERROR">/var/log/arched3.log</Logger>
    </Server>
    <ModuleManager>
        <Path>/usr/local/lib/arc/</Path>
    </ModuleManager>
    <Plugins><Name>mcctcp</Name></Plugins>
    <Plugins><Name>mcctls</Name></Plugins>
    <Plugins><Name>mcchttp</Name></Plugins>
    <Plugins><Name>mccsoap</Name></Plugins>
    <Chain>
        <Component name="tcp.service" id="tcp">
            <next id="tls"/>
            <tcp:Listen>
<tcp:Port>60010</tcp:Port>
<tcp:Version>4</tcp:Version>
    </tcp:Listen>
        </Component>
        <Component name="tls.service" id="tls">
            <next id="http"/>
            <KeyPath>/etc/grid-security/hostkey.pem</KeyPath>
            <CertificatePath>/etc/grid-security/hostcert.pem</CertificatePath>
            <CACertificatesDir>/etc/grid-security/certificates</CACertificatesDir>
        </Component>
        <Component name="http.service" id="http">
            <next id="soap">POST</next>
        </Component>
        <Component name="soap.service" id="soap">
            <next id="plexer"/>
        </Component>
        <Plexer name="plexer.service" id="plexer">
            <next id="shepherd">^/Shepherd$</next>
        </Plexer>
        <Service name="pythonservice" id="shepherd">
            <ClassName>storage.shepherd.shepherd.ShepherdService</ClassName>
            <ServiceID>https://storage3:60010/Shepherd</ServiceID>
            <CheckPeriod>20</CheckPeriod>
            <MinCheckInterval>0.1</MinCheckInterval>
            <CreatingTimeout>600</CreatingTimeout>
            <StoreClass>arcom.store.cachedpicklestore.CachedPickleStore</StoreClass>
            <StoreCfg><DataDir>/var/spool/arc/shepherd_data3</DataDir></StoreCfg>
            <BackendClass>storage.shepherd.hardlinkingbackend.HopiBackend</BackendClass>
            <BackendCfg>
                <DataDir>/var/spool/arc/shepherd_store3</DataDir>
                <TransferDir>/var/spool/arc/shepherd_transfer3</TransferDir>
                <TURLPrefix>http://storage3:60011/hopi/</TURLPrefix>
            </BackendCfg>
            <LibrarianURL>https://storage:60000/Librarian</LibrarianURL>
            <BartenderURL>https://storage:60000/Bartender</BartenderURL>
            <ClientSSLConfig FromFile="/etc/arc/clientsslconfig.xml" />
        </Service>
    </Chain>
    <Chain>
        <Component name="tcp.service" id="tcp2">
            <next id="http2"/>
            <tcp:Listen>
<tcp:Port>60011</tcp:Port>
<tcp:Version>4</tcp:Version>
    </tcp:Listen>
        </Component>
        <Component name="http.service" id="http2">
            <next id="plexer2">GET</next>
            <next id="plexer2">PUT</next>
        </Component>
        <Plexer name="plexer.service" id="plexer2">
            <next id="hopi">^/hopi/</next>
        </Plexer>
        <Service name="hopi" id="hopi">
            <DocumentRoot>/var/spool/arc/shepherd_transfer3</DocumentRoot>
            <SlaveMode>1</SlaveMode>
        </Service>
    </Chain>
</ArcConfig>
```

This configuration has two separate ports, one with TLS security and a Shepherd service, and the other without TLS security and with the Hopi service. Run this server as well:

```
$ sudo /usr/local/sbin/arched -c /etc/arc/storage_service_3.xml -f
```

Now when checking the file again there should be no change at all:

```
$ arc_storage_cli stat /orange
  locations
    https://storage:60000/Shepherd 315d59d3-50cd-1a10-02e6-8e6a528473c3: alive
    https://storage2:60006/Shepherd c1334c09-5d3d-983b-2081-87fe60d4764a: alive
```

Stop the server on storage2, and watch what happens with the file:

```
$ arc_storage_cli stat /orange
  locations
    https://storage:60000/Shepherd 315d59d3-50cd-1a10-02e6-8e6a528473c3: alive
    https://storage2:60006/Shepherd c1334c09-5d3d-983b-2081-87fe60d4764a: offline
```

```
$ arc_storage_cli stat /orange
  locations
    https://storage:60000/Shepherd 315d59d3-50cd-1a10-02e6-8e6a528473c3: alive
    https://storage2:60006/Shepherd c1334c09-5d3d-983b-2081-87fe60d4764a: offline
    https://storage3:60010/Shepherd 421aa939-81fa-5cd9-7c2f-2d5f99a98be1: creating

$ arc_storage_cli stat /orange
  locations
    https://storage:60000/Shepherd 315d59d3-50cd-1a10-02e6-8e6a528473c3: alive
    https://storage2:60006/Shepherd c1334c09-5d3d-983b-2081-87fe60d4764a: offline
    https://storage3:60010/Shepherd 421aa939-81fa-5cd9-7c2f-2d5f99a98be1: alive
```

A new replica has been created on the third server to maintain the needed number of 2.

Now, start the services on `storage2` again, and see what is happening with the file:

```
$ arc_storage_cli stat /orange
  locations
    https://storage:60000/Shepherd 315d59d3-50cd-1a10-02e6-8e6a528473c3: alive
    https://storage2:60006/Shepherd c1334c09-5d3d-983b-2081-87fe60d4764a: alive
    https://storage3:60010/Shepherd 421aa939-81fa-5cd9-7c2f-2d5f99a98be1: alive

$ arc_storage_cli stat /orange
  locations
    https://storage:60000/Shepherd 315d59d3-50cd-1a10-02e6-8e6a528473c3: alive
    https://storage2:60006/Shepherd c1334c09-5d3d-983b-2081-87fe60d4764a: thirdwheel
    https://storage3:60010/Shepherd 421aa939-81fa-5cd9-7c2f-2d5f99a98be1: alive

$ arc_storage_cli stat /orange
  locations
    https://storage:60000/Shepherd 315d59d3-50cd-1a10-02e6-8e6a528473c3: alive
    https://storage3:60010/Shepherd 421aa939-81fa-5cd9-7c2f-2d5f99a98be1: alive
```

One of the replicas has been removed to maintain the needed number of 2.

4 Make the A-Hash service replicated

Now the deployment is as following: One machine has the A-Hash, Librarian and Bartender services, and all three machines have a Shepherd and a storage element service. The A-Hash service stores all the metadata of all the files and collections, so it is critical. If the first machine goes offline then the whole system dies. However, it is possible to deploy the A-Hash service on all the machines and still have one consistent metadata database. The replicated version of the A-Hash uses the Berkeley DB to provide a distributed database.

In the following, Berkeley DB is installed as root (the `--prefix` option of `configure` can be used to install it somewhere else than the default location). This needs to be done on all the machines.

Get the latest Berkeley DB and its patches:

```
$ wget http://download.oracle.com/berkeley-db/db-4.7.25.tar.gz
$ wget http://www.oracle.com/technology/products/berkeley-db/db/update/4.7.25/patch.4.7.25.1
$ wget http://www.oracle.com/technology/products/berkeley-db/db/update/4.7.25/patch.4.7.25.2
$ wget http://www.oracle.com/technology/products/berkeley-db/db/update/4.7.25/patch.4.7.25.3
```

Extract it and apply the patches:

```
$ tar zxf db-4.7.25.tar.gz
$ cd db-4.7.25
$ patch -p 0 < ../patch.4.7.25.1
$ patch -p 0 < ../patch.4.7.25.2
$ patch -p 0 < ../patch.4.7.25.3
```

Configure, make and install it (and if needed, use the `--prefix` option for `configure`)

```
$ cd build_unix/
$ ../dist/configure
```

Now compile and install it:

```
$ make
$ sudo make install
```

Next, get the python wrapper for the Berkeley DB:

```
$ wget http://pypi.python.org/packages/source/b/bsddb3/bsddb3-4.7.5.tar.gz
```

Install it to the default location:

```
$ tar zxf bsddb3-4.7.5.tar.gz
$ cd bsddb3-4.7.5
$ sudo python setup.py install
```

If Berkeley DB was installed in a non-default location, or to install the python wrapper to somewhere else than the default:

```
$ BERKELEYDB_DIR=/path/to/berkeleydb python setup.py install --prefix=/where/to/install
```

Test the installation with the `test.py` included in the directory of the python wrapper for the Berkeley DB:

```
$ sudo python test.py
[...]
Ran 383 tests in 29.858s

OK
```

Configure the first server (`storage`). Here is the AHash part of the config:

```
<Service name="pythonservice" id="ahash">
    <ClassName>storage.ahash.ahash.AHashService</ClassName>
    <AHashClass>storage.ahash.replicatedahash.ReplicatedAHash</AHashClass>
    <LocalDir>/var/spool/arc/ahash_data</LocalDir>
    <MyURL>https://storage:60000/AHash</MyURL>
    <OtherURL>https://storage2:60006/AHash</OtherURL>
    <Priority>50</Priority>
    <ClientSSLConfig FromFile='/etc/arc/clientsslconfig.xml'/>
</Service>
```

In this config the second server is specified as a peer. Configure the second server (`storage2`) as well:

```
<Plexer name="plexer.service" id="plexer">
    <next id="ahash">^/AHash$</next>
    <next id="shepherd">^/Shepherd$</next>
    <next id="byteio">^/byteio/</next>
```

```
</Plexer>
<Service name="pythonservice" id="ahash">
    <ClassName>storage.ahash.ahash.AHashService</ClassName>
    <AHashClass>storage.ahash.replicatedahash.ReplicatedAHash</AHashClass>
    <LocalDir>/var/spool/arc/ahash_data2</LocalDir>
    <MyURL>https://storage2:60006/AHash</MyURL>
    <OtherURL>https://storage:60000/AHash</OtherURL>
    <Priority>50</Priority>
    <ClientSSLConfig FromFile='/etc/arc/clientsslconfig.xml'/>
</Service>
```

In this config the first server is configured as a peer. So the two together will be able to form a replicated network with two nodes. Then configure the third one (storage3):

```
<Plexer name="plexer.service" id="plexer">
    <next id="ahash">^/AHash$</next>
    <next id="shepherd">^/Shepherd$</next>
</Plexer>
<Service name="pythonservice" id="ahash">
    <ClassName>storage.ahash.ahash.AHashService</ClassName>
    <AHashClass>storage.ahash.replicatedahash.ReplicatedAHash</AHashClass>
    <LocalDir>/var/spool/arc/ahash_data3</LocalDir>
    <MyURL>https://storage3:60010/AHash</MyURL>
    <OtherURL>https://storage:60000/AHash</OtherURL>
    <Priority>50</Priority>
    <ClientSSLConfig FromFile='/etc/arc/clientsslconfig.xml'/>
</Service>
```

Here the first server is specified as the peer.

It is important to make a fresh start on all the servers, and remove the existing database, which was created by a different store:

```
$ sudo rm -r /var/spool/arc/*
```

Now, start the first server. It will complain that it cannot find its peer:

```
ERROR connecting to https://storage2:60006/AHash
[2009-03-24 10:34:18] [Arc.Storage.ReplicatedAHash] [ERROR] [10827/156026112]
    failed to send to 2 of [1, 2]
[2009-03-24 10:34:20] [Arc.Storage.ReplicatedAHash] [ERROR] [10827/158713512]
    Couldn't run election
```

With only one node of the replicated A-Hash the system is not ready to work, and trying to use it will result in an error message:

```
$ arc_storage_cli makeCollection /
- Calling the Bartender's makeCollection method...
- done in 0.55 seconds.
Creating collection '/': failed to create new librarian entry
```

Start the second server. It may first print some error messages, but then both servers should be quiet, and the system is ready for use. Now, upload a file, and set the number of needed replicas to 3:

```
$ arc_storage_cli makeCollection /
- Calling the Bartender's makeCollection method...
- done in 1.16 seconds.
Creating collection '/': done
```

```
$ arc_storage_cli put orange /
- The size of the file is 7 bytes
- Calculating md5 checksum...
- done in 0.0214 seconds, md5 checksum is 6aefd2842be62cd470709b27aedc7db7
- Calling the Bartender's putFile method...
- done in 2.22 seconds.
- Got transfer URL: http://storage:60001/hopi/3f903c50-0424-2891-5730-521cc00d96dc
- Uploading from 'orange'
    to 'http://storage:60001/hopi/3f903c50-0424-2891-5730-521cc00d96dc' with http...
    1 s:        0.0 kB       0.0 kB/s       0.0 kB/s    . . .
- done in 1.0293 seconds.
'orange' (7 bytes) uploaded as '/orange'.
$ arc_storage_cli modify /orange set states neededReplicas 3
- Calling the Bartender's modify method...
- done in 0.74 seconds.
set
$ arc_storage_cli stat /orange
[...]
  locations
    https://storage:60000/Shepherd f4ad8530-354d-5066-04ae-c54c94fd525b: alive
[...]
zsombor@storage:~$ arc_storage_cli stat /orange
[...]
  locations
    https://storage:60000/Shepherd f4ad8530-354d-5066-04ae-c54c94fd525b: alive
    https://storage2:60006/Shepherd 97edebb5-e72d-098a-55a7-14595f668770: alive
[...]
```

It cannot create 3 replicas, because the third server is not running. Start it. After a minute or two, the file has three replicas:

```
$ arc_storage_cli stat /orange
[...]
  locations
    https://storage:60000/Shepherd f4ad8530-354d-5066-04ae-c54c94fd525b: alive
    https://storage3:60010/Shepherd 3c55924b-73ad-c226-0f4a-4c056c967314: alive
    https://storage2:60006/Shepherd 97edebb5-e72d-098a-55a7-14595f668770: alive
[...]
```

The fact that this file has three replicas is not at all related to the number of AHashes, but to the number of storage elements (Shepherds and storage element services), so this is no different then what was seen in the previous section. The benefit of having more then one AHashes in a replicated way is that if one of the machines goes offline, the AHash is still working. However, with the setup above, if the first server is shutdown, the AHash will still work but there is only one instance of the other two important service: the Librarian and the Bartender, so the whole system will still be unusable. Hence, more Librarians and Bartenders needs to be added to the system. The next section explains how.

5 Additional Bartenders and Librarians

In this section, a Bartender and a Librarian is added to each server. The first server does not need modification now. For the other two, the `Plexer` section needs to be altered, the twos services must be added, and the Shepherd must be modified to use the server's own Bartender and Librarian for replication and heartbeats. Modify the config of the second server as follows: (`[...]` indicates unchanged parts)

```
[...]
<Plexer name="plexer.service" id="plexer">
```

```
    <next id="ahash">^/AHash$</next>
    <next id="shepherd">^/Shepherd$</next>
    <next id="librarian">^/Librarian$</next>
    <next id="bartender">^/Bartender$</next>
    <next id="byteio">^/byteio/</next>
</Plexer>
<Service name="pythonservice" id="librarian">
    <ClassName>storage.librarian.librarian.LibrarianService</ClassName>
    <AHashURL>https://storage2:60006/AHash</AHashURL>
    <HeartbeatTimeout>30</HeartbeatTimeout>
    <CheckPeriod>20</CheckPeriod>
    <ClientSSLConfig FromFile="/etc/arc/clientsslconfig.xml" />
</Service>
<Service name="pythonservice" id="bartender">
    <ClassName>storage.bartender.bartender.BartenderService</ClassName>
    <LibrarianURL>https://storage2:60006/Librarian</LibrarianURL>
    <ClientSSLConfig FromFile="/etc/arc/clientsslconfig.xml" />
</Service>
[...]
<Service name="pythonservice" id="shepherd">
[...]
    <LibrarianURL>https://storage2:60006/Librarian</LibrarianURL>
    <BartenderURL>https://storage2:60006/Bartender</BartenderURL>
[...]
```

And the third server:

```
[...]
<Plexer name="plexer.service" id="plexer">
    <next id="ahash">^/AHash$</next>
    <next id="shepherd">^/Shepherd$</next>
    <next id="librarian">^/Librarian$</next>
    <next id="bartender">^/Bartender$</next>
</Plexer>
<Service name="pythonservice" id="librarian">
    <ClassName>storage.librarian.librarian.LibrarianService</ClassName>
    <AHashURL>https://storage3:60010/AHash</AHashURL>
    <HeartbeatTimeout>30</HeartbeatTimeout>
    <CheckPeriod>20</CheckPeriod>
    <ClientSSLConfig FromFile="/etc/arc/clientsslconfig.xml" />
</Service>
<Service name="pythonservice" id="bartender">
    <ClassName>storage.bartender.bartender.BartenderService</ClassName>
    <LibrarianURL>https://storage3:60010/Librarian</LibrarianURL>
    <ClientSSLConfig FromFile="/etc/arc/clientsslconfig.xml" />
</Service>
[...]
<Service name="pythonservice" id="shepherd">
[...]
    <LibrarianURL>https://storage3:60010/Librarian</LibrarianURL>
    <BartenderURL>https://storage3:60010/Bartender</BartenderURL>
[...]
```

client.xml must be modified to add all the other Bartenders:

```
$ cat ~/.arc/client.xml
<ArcConfig>
  <KeyPath>/home/zsombor/.arc/userkey.pem</KeyPath>
  <CertificatePath>/home/zsombor/.arc/usercert.pem</CertificatePath>
  <CACertificatesDir>/etc/grid-security/certificates</CACertificatesDir>
  <BartenderURL>https://storage:60000/Bartender</BartenderURL>
  <BartenderURL>https://storage2:60006/Bartender</BartenderURL>
  <BartenderURL>https://storage3:60010/Bartender</BartenderURL>
</ArcConfig>
```

Now, all three servers can be started. The replicated A-Hash sometimes prints great amount of error messages, but if it stops printing them, then everything is OK. Now the system can be used as intended. There is a flag for the `arc_storage_cli` which makes it very verbose. You can check the used Bartender with it:

```
$ arc_storage_cli -v stat /orange
- The URL of the Bartender(s): https://storage:60000/Bartender,
    https://storage2:60006/Bartender, https://storage3:60010/Bartender
- The key file: /home/zsombor/.arc/userkey.pem
- The cert file: /home/zsombor/.arc/usercert.pem
- The CA dir: /etc/grid-security/certificates
- Calling the Bartender's stat method...
- trying https://storage:60000/Bartender
[...]
- done in 1.27 seconds.
'/orange': found
```

Now, stop the first server. The system will be unaccessible for a few seconds while the other two A-Hash figure out what is happening, but after that it will work again:

```
$ arc_storage_cli -v stat /orange
- The URL of the Bartender(s): https://storage:60000/Bartender,
    https://storage2:60006/Bartender, https://storage3:60010/Bartender
- The key file: /home/zsombor/.arc/userkey.pem
- The cert file: /home/zsombor/.arc/usercert.pem
- The CA dir: /etc/grid-security/certificates
- Calling the Bartender's stat method...
- trying https://storage:60000/Bartender
[...]
- (-1, "Wrong status from server.\nMaybe wrong URL: 'https://storage:60000/Bartender'")
- trying https://storage2:60006/Bartender
[...]
- done in 0.45 seconds.
'/orange': found
[...]
```

If the first server is started again, after a few seconds the system will work again with the three fully-functional node.

```
zsombor@storage:~$ arc_storage_cli -v stat /orange
- The URL of the Bartender(s): https://storage:60000/Bartender,
    https://storage2:60006/Bartender, https://storage3:60010/Bartender
- The key file: /home/zsombor/.arc/userkey.pem
- The cert file: /home/zsombor/.arc/usercert.pem
- The CA dir: /etc/grid-security/certificates
- Calling the Bartender's stat method...
- trying https://storage:60000/Bartender
[...]
- done in 0.49 seconds.
'/orange': found
```

```
  states
    checksumType: md5
    neededReplicas: 3
    size: 7
    checksum: 6aefd2842be62cd470709b27aedc7db7
  entry
    owner: /DC=eu/DC=KnowARC/O=knowarc/CN=billy
    GUID: 0080d8be-1f6f-2fc7-ad71-732f899326c8
    type: file
  parents
    0/orange: parent
  locations
    https://storage3:60010/Shepherd 03312843-d3b3-2d15-4732-ad673df958ae: alive
    https://storage2:60006/Shepherd b0057765-4532-d0b4-fded-182499a91450: alive
    https://storage:60000/Shepherd 532ec0f8-dd83-297b-d7b6-dc5dacea34e0: alive
  timestamps
    created: 1237887474.95
```

6 Accessing external storage solutions

If you have permissions to access some third-party storage solutions (e.g. dCache via the GridFTP protocol), you can create a mount point within the namespace of the ARC storage system which points to a third-party URL, and then you can use the client interface of the ARC storage to access the third-party storage as well. This is achieved by a module of the Bartender, called the 'Gateway'. Currently only GridFTP access is supported, and for that the Globus libraries needs to be installed. Next, ARC needs to be recompiled for GridFTP support. The output of the `configure` script should look like this:

```
    Available third-party features:

    RLS:                yes
    GridFTP:            yes
    LFC:                no
    RSL:                yes
    SAML:               yes
    MYSQL CLIENT LIB:   no
    gSOAP:              yes

    Included components:
    A-Rex service:      no
    ISI service:        no
    CHARON service:     no
    HOPI service:       yes
    SCHED service:      no
    STORAGE service:    yes
    PAUL service:       no
    SRM client (DMC):   yes
    GSI channel (MCC):  yes
```

After compilation and installation the `arcls` tool can be used to list the content of a third-party storage. The user Billy can see if has access to this directory:

```
$ arcls gsiftp://sal1.uppmax.uu.se:2811/pnfs/uppmax.uu.se/data
data1
data2
other
```

Since he can see some files in there, so he have permissions to list that directory.

Now, configure the Bartenders on all three machines to use the Gateway module:

```
<Service name="pythonservice" id="bartender">
    <ClassName>storage.bartender.bartender.BartenderService</ClassName>
    <LibrarianURL>[...]</LibrarianURL>
    <ProxyStore>/var/spool/arc/proxy_store</ProxyStore>
    <GatewayClass>storage.bartender.gateway.gateway.Gateway</GatewayClass>
    <GatewayCfg>
        <ProxyStore>/var/spool/arc/proxy_store</ProxyStore>
        <CACertificatesDir>/etc/grid-security/certificates</CACertificatesDir>
    </GatewayCfg>
    <ClientSSLConfig FromFile="/etc/arc/clientsslconfig.xml" />
</Service>
```

Next, start all three servers. Then create a mount point with the Logical Name `/salmount` which points to this URL:

```
$ arc_storage_cli makemount /salmount \
    gsiftp://sal1.uppmax.uu.se:2811/pnfs/uppmax.uu.se/data
- Calling the Bartender's makeMountpoint method...
- done in 3.07 seconds.
Creating mountpoint '/salmount': done

$ arc_storage_cli stat /salmount
- Calling the Bartender's stat method...
- done in 0.47 seconds.
'/salmount': found
  states
    closed: 0
  entry
    owner: /DC=eu/DC=KnowARC/O=knowarc/CN=billy
    type: mountpoint
    GUID: b21d9d22-e885-502a-ee0e-bd0a7b967543
  parents
    0/salmount: parent
  mountpoint
    externalURL: gsiftp://sal1.uppmax.uu.se:2811/pnfs/uppmax.uu.se/data
  timestamps
    created: 1237990168.09
```

List the content of this mountpoint:

```
$ arc_storage_cli list /salmount- Calling the Bartender's list method...
- done in 0.49 seconds.
'/salmount': Your proxy cannot be found. Please delegate your credentials!
```

OK, First you need to delegate your credentials with the `credentialsDelegation` method.

```
$ arc_storage_cli cre
- Calling the Bartender's credentialsDelegation method...
[...]
- done in 0.42 seconds.
Successful delegation.
Proxy ID: eNCMDmi8i6YnPSAtDmVmuSEmABFKDmABFKDmOzKKDmFBFKDmeOzdBo
```

Then try to list the mount point again:

```
$ arc_storage_cli list /salmount
- Calling the Bartender's list method...
- done in 3.19 seconds.
'/salmount': collection
  data1  <unknown>
  data2  <unknown>
  other  <unknown>
```

Now, download a file:

```
$ arc_storage_cli get /salmount/data1 /tmp
- Calling the Bartender's getFile method...
- done in 3.56 seconds.
- Got transfer URL: gsiftp://sal1.uppmax.uu.se:2811/pnfs/uppmax.uu.se/data/data1
- Downloading from 'gsiftp://sal1.uppmax.uu.se:2811/pnfs/uppmax.uu.se/data/data1'
    to '/tmp/data1' with gridftp...
   2 s:        0.0 kB       0.0 kB/s       0.0 kB/s    . . .
   3 s:        1.3 kB       0.4 kB/s       0.4 kB/s    . . .
- done in 2.7828 seconds.
'/salmount/data1' (1290 bytes) downloaded as '/tmp/data1'.
```

7 Using the ARC DMC

ARC has its own multi-protocol storage client tools which provide a unified way to access different kind of storage. For all the supported types there is a DMC module. For Chelonia, the ARC DMC module is used. If it is compiled and installed, then you can use the `arcls`, `arccp`, `arcrm`, etc. tools with the `arc://` protocol to access the ARC storage system. You can specify the URL of the Bartender with the `BartenderURL` URL option, or you can have a Bartender URL configured in the `client.xml` user config.

```
$ arcls arc:///
$ arccp orange arc:///orange
$ arcls arc:///
orange
$ arcls arc:///?BartenderURL=https://storage2:60006/Bartender
orange
$ arc_storage_cli list /
- Calling the Bartender's list method...
- done in 0.56 seconds.
'/': collection
  orange <file>
$ arc_storage_cli makeCollection /fruits
- Calling the Bartender's makeCollection method...
- done in 2.38 seconds.
Creating collection '/fruits': done
$ arccp arc:///orange arc:///fruits/orange
$ arcls arc:///
orange
fruits
$ arcls arc:///fruits
orange
$ arccp arc:///orange /tmp/orange
$ cat /tmp/orange
orange
$ arcrm arc:///orange
$ arcls arc:///
fruits
```

8 Using the FUSE module

Besides the `arc_storage_cli` tool you can access the storage system with the use of a FUSE module. The following steps installs the ARC storage FUSE module:

- Install ARC1 storage system.
- Install `fuse >= 2.7.3` and `fuse-python >= 0.2`:

  ```
  $ sudo aptitude install fuse-python
  ```

- Set up `~/.arc/client.xml`:

  ```
  $ cat ~/.arc/client.xml
  <ArcConfig>
    <KeyPath>/home/<username>/.arc/userkey.pem</KeyPath>
    <CertificatePath>/home/<username>/.arc/usercert.pem</CertificatePath>
    <CACertificatesDir>/etc/grid-security/certificates</CACertificatesDir>
    <BartenderURL>https://localhost:60000/Bartender</BartenderURL>
  </ArcConfig>
  ```

- Create a mountpoint and mount arcfs.py:

  ```
  $ pwd
  /home/me/arc
  $ ln -s arc1/src/services/storage/fuse/arcfs.py arcfs.py
  $ mkdir mnt
  $ python arcfs.py ./mnt
  ```

  This mounts ARCFS in `/home/me/arc/mnt`, creates directory `/home/me/arc/fuse_transfer` and a log file `/home/me/arc/arcfsmessages`.

- Make a collection:

  ```
  $ mkdir -p mnt/home/me
  $ stat mnt/home/me
    File: ‘mnt/home/me’
    Size: 0               Blocks: 0          IO Block: 4096   directory
  Device: 19h/25d Inode: 3          Links: 2
  Access: (0755/drwxr-xr-x)  Uid: (  500/  me)   Gid: (  100/   users)
  Access: 1970-01-01 01:00:00.000000000 +0100
  Modify: 1970-01-01 01:00:00.000000000 +0100
  Change: 1970-01-01 01:00:00.000000000 +0100
  ```

  Note that so far there is no security handling implemented, so that all files have mode 0644, all collections have mode 0755, everything is owned by the user that mounts the system.

- Create some entries in your collection:

  ```
  $ cd mnt/home/me
  $ emacs -nw fish
  catfish
  $ cat fish
  catfish
  $ cp fish fish2
  $ ls -la
  drwxr-xr-x 5 me users  0 1970-01-01 01:00 .
  drwxr-xr-x 3 me users  0 1970-01-01 01:00 ..
  -rw-r--r-- 1 me users  8 1970-01-01 01:00 fish
  -rw-r--r-- 1 me users 28 1970-01-01 01:00 fish~
  -rw-r--r-- 1 me users  8 1970-01-01 01:00 fish2
  ```

- You probably want to remove that anoying emacs backup file

  ```
  $ rm fish~
  $ ls
  fish  fish2
  ```

- Maybe you want fish2 in a separate collection

  ```
  $ mkdir sea_creatures
  $ mv fish2 sea_creatures
  ```

- Maybe you're not happy with the collection name

  ```
  $ mv sea_creatures creatures
  $ ls creatures
  fish2
  ```

- Moving the collection out of ARC storage:

  ```
  $ cd ../../..
  $ mv mnt/home/me/creatures .
  ```

- The collection is no longer in the ARC storage:

  ```
  $ find mnt/home/me/
  mnt/home/me/
  mnt/home/me/fish
  ```

- Unmounting ARCFS; To unmount, refer to fuse documentation. Usually you can do

  ```
  $ fusermount -u mnt
  ```

  If this for some reason this doesn't work, you can do

  ```
  $ sudo umount -f mnt
  ```

  or even

  ```
  $ sudo umount -l mnt
  ```

  Now, remounting the system, everything should still be there:

  ```
  $ python arcfs.py ./mnt

  $ find mnt/home/me/
  mnt/home/me/
  mnt/home/me/fish

  $ fusermount -u mnt
  ```