ARC Storage

The storage solution of the new ARC middleware will contain these components:

- the distributed Data Catalogue,
- several Storage Managers,
- ARC Storage Elements,
- gateways to third-party storage elements (TSE),
- gateways to third-party storage systems (TSS),
- a distributed Maintainer Service,
- client.

Data Catalogue

The Data Catalogue (or just 'Catalog') is a database which stores data about Files and group these Files into Collections. Each File and Collection has a globally unique ID called 'GUID'. So each Catalog entry can be unambiguously referenced by using its GUID.

Collections

A Collection is a list of Files and other Collections, which are in parent-children relationships forming a tree-hierarchy. Each entry has a name which is only valid within this Collection, and it is unique within the Collection. Each entry is referenced by its GUID. So a Collection is a list of name-GUID pairs, plus there are also some well-defined metadata associated with the Collection in the form of key-value pairs such as timestamp, owner, access control list, etc., and any other arbitrary metadata can be stored with it.

Files

A File entry contains a couple of attributes such as size, timestamp, owner, list of access control rules, etc. It has a list of locations of the replicas, and the state of each replicas. Other arbitrary metadata can also be associated with a File entry.

Externals

There is one more type of Catalog entries called External, which is a link to an external service which has a well-defined interface, and can handle a subtree of the global namespace. It could be another catalog, or a third-party storage system, etc.

In ARC Storage system there is one global Data Catalogue which should be implemented in a distributed way for providing high-availability.

Logical Name (LN)

Each file and collection has a GUID which is globally unique, so they can be unambiguously referred using this GUID, that's why a single GUID is a Logical Name itself (if we put a slash on the end of it: '1234/')

In a collection each entry has a name, and this entry can be a sub-collection, in which there are files and sub-sub-collections, etc. e.g. if we have a collection with GUID '1234', and in it there is a collection called 'abc', and in 'abc' there is another collection called 'def', and in 'def' there is the file called 'ghi', then we can refer to this file as '/abc/def/ghi', if we know the GUID of the starting collection, so let's prefix the path with it: '1234/abc/def/ghi'. This is the Logical Name of that file. If there is a well-known system-wide root collection (its GUID could be e.g. '0'), then if a LN starts with no GUID prefix, it is implicitly prefixed with the GUID of this well-known root collection, e.g. '/what/ever' means '0/what/ever'.

The syntax of a Logical Name (LN): [<GUID>]/[<path>]

If a client wants to find the file called '/what/ever', the client knows where to start the search, it knows the GUID of the root collection. The root collection knows the GUID of 'what', and the (sub-)collection 'what' knows the GUID of 'ever'. If the GUID of this file is '5678', and somebody makes another entry in collection '/what' (= '0/what') with name 'else' and GUID '5678', then the '/what/else' LN points to the same file as '/what/ever', so it's a hard link.

Each VO should create a VO-wide root collection, and put it in the generic root collection, e.g. if a VO called 'vo1' creates a collection called 'vo1' as a sub-collection of the root collection (which has the guid '0'), then it can be refered as '0/vo1' or just '/vo1'. Then this VO can create some files, and put them in this '/vo1' collection, e.g. '/vo1/file1', etc. Or sub-collections, e.g. '/vo1/col1', '/vo1/col2/file3', etc. For this the VO does not need the install any service. These files and collection s can be created using the storage management layer (using one of the Storage Managers).

Storage Managers

Clients can access the storage system only through a Storage Manager. If a client wants to create a collection, upload or download a file, the first step is to connect a Storage Manager. The Storage Manager then resolves Logical Names and gets metadata using the catalog, initiates file transfers on some storage elements, and gives some assertions (some kind of certificate) to the client, which allows the client to actually do the file transfer from/to a storage element. So the data transfer itself does not go through the Storage Manager, it is performed over a direct link between a storage element and the client.

In case of upload the Storage Manager is capable of choosing appropiate Storage Elements using pluggable scheduling algorithms.

ARC Storage Elements

The file replicas are stored on the Storage Elements.

The hierarchy of files on an ARC Storage Element has nothing to do with the the hierarchy of collections, or Logical Names. The interface of an ARC Storage Element works with SURLs and TURLs. The SURL is a system-wide unique ID of a file on a Storage Element. If you have a SURL and an assertion which authorize you, you can turn to the Storage Element specified by the SURL itself, and the Storage Element will recognize the SURL as a file it actually stores. Then the Storage Element provides a way to download that file (it generates a Transfer URL, TURL, which can be an http(s) or gridftp, etc.) If you want to upload a new file to the storage system, then you turn to a Storage Manager, which chooses a Storage Element, and initiates the transfer. Then a new SURL is created, and the Storage Element will provide a way to upload the file, which later can be referred using this new SURL. So it does not matter which collection this file is part of, and it does not matter how the Storage Element internally organize these files. The file has an SURL, and by referencing that, it can be downloaded.

The SURL refers to the Storage Element itself, and the file on the Storage Element, e.g. https://storage.element.eu:12345/SE?ocrh6kb43d8j

Third-party Storage Elements (TSEs)

There is a possibility to use third-party software as Storage Element.

If we already have a working storage solution with lots of free space, and we do not want to integrate the existing data on this storage solution to the ARC Storage, but to use the free space on it, then it could be attached to the ARC Storage as a Third-party Storage Element. We need to install a TSE Gateway service somewhere, which is wired into this third-party software, and provide a well-defined interface to the Storage Managers. E.g. if we have an ordinary FTP server, and we have a TSE Gateway which is able to initiate transfer on this FTP server and to generate some kind of SURLs,

then this FTP server can be used as a Third-party Storage Element. We cannot access the data which are already on this FTP server, and if someone put a file on it directly, not through one of the ARC Storage Managers, this file won't be accessible by the ARC storage. If we have two FTP servers to attach, we should install two TSE Gateways, one for each. Or we can install more than one for each for high-availability, but one TSE Gateway is explicitly wired to one physical storage.

Third-party Storage Systems (TSSs)

If we want to attach working third-party storage systems full of data as a Third-party Storage System, we need a TSS Gateway. This method is very similar to mounting an external file system to a mount point of our own file system. This TSS Gateway has to be very sophisticated to be as transparent as possible. e.g. we have an external storage solution having its own namespace and protocol. There is a file on this storage, which can be referred as '/moz/art' in the namespace of the third-party storage. On our ARC Storage we have a Collection called '/vo/mnt'. The mounting is done by insert ing an External entry into the Catalog which links to the TSS Gateway installed for this third-party storage, so we could have a mount point at '/vo/mnt/ext1'. After this, the content of the third-party storage system is available through the ARC Storage using LNs, e.g. '/vo/mnt/ext1/moz/art'.

When a client wants to get this file, it turns to a Storage Manager with this LN. The Storage Manager uses the Catalog to resolve this LN, but the Catalog is not able to resolve it completely, because when it reaches '/vo/mnt/ext1', it turns out, that it is an External mount point, with the address of the TSS Gateway. Then the Storage Manager turns to the Gateway, which gets the remaining part of LN ('/moz/art'), connects to the third-party storage system, and initiate

file transfer, etc. If a client wants to upload to '/vo/mnt/ext1/wag/ner', then the TSS Gateway will get '/wag/ner'.

It could be very difficult with a third-party storage system to initiate transfer, to create assertions which behaves the same as the assertions of native ARC Storage Elements. The ARC Catalog has no metadata about anything on these mounted storage systems, because it stores only the mount point. These files and directories on these third-party storage systems have no GUID, they cannot be part of a collection (on a linux file system you cannot create a hard link to a file in a mounted external file system). There is no replication of these data, it is assumed that the third-party storage system is fault-tolerant. (The replication is on the level of SURLs, mounting of a third-party storage system is on the level of LNs.)

The main purpose of Third-party Storage System Gateways is to integrate external existing storage system with a lots of data to the global namespace of the ARC Storage, and to make them available through the same interface as the 'native' data. If a VO has its well-known third-party storage solution which is permanently mounted to e.g. '/vo/data', and a user has a '/home/jim' directory on this external storage, then the user can specify this as stage-out destination of a job, e.g. '/vo/data/home/jim/myjob123', and when the job is done, it uploads its results through the ARC Storage Management layer to this external storage, which later can be directly accessed by the user.

A special TSS Gateway is the Sharing Service.

Sharing Service

If we want to share a local directory full of files to the users of the ARC Storage, we could install a Sharing Service to the machine, and then mount it somewhere to the global namespace. e.g. if we have a directory on our machine 'share', and there is a file 'share/dir/file' in that, and we install this Sharing Service, set it up to share this 'share' directory, then mount it to the global namespace at '/myvo/home/joe/ext', then anyone who are allowed can access the file through the ARC storage with the LN '/myvo/home/zsombor/ext/dir/file'. This is preferably a read-only share, but if it is not, then a user can put a file to '/myvo/home/zsombor/ext/file2' which after uploading appears on our local machine. A Sharing Service is actually a TSS Gateway: the Catalog knows nothing about the files and directories on this shared directory, these files has no GUID, etc.

By using this concept we can easily create a read-only share for the working directory of a running job to make the job's intermediate results available through the ARC storage before the job is completed and the job manager uploads the results to the storage.

The Maintainer Service

The Maintainer Service is responsible for ensuring each file has at least as many valid replicas as needed. It has a list of files to be checked. This candidate list is continuously appended by walking the namespace of the storage or by querying the Catalog using GUID ranges, so each file in the Catalog will be periodically checked. This list can be appended manually, e.g. by the Storage Manager, when a new file is created. These manually appended files has larger priority. Each file has a checksum (e.g. CRC) which can be used to check the validity of replicas. Each replica has a state, e.g. valid, invalid, offline, yet-to-be-uploaded, flagged-for-deletition, etc. If a file has at least one valid replica, but has less valid replicas than needed, the Maintainer Service creates more replicas, or asks a Storage Manager to do it. Which of these two is still an open question, no decisions yet.

The Maintainer Service must be distributed for load-balancing, and the nodes should maintain a common candidate list, but it does not seem fatal if a file accidentally is checked more then once, so the consistency of the candidate list is not critical.

Client

There should be a client library for the ARC Storage which provides higher level functionality than a Storage Manager itself, e.g. recursive download and upload of directory structures, which means the client fetch a recursive listing of the source subtree, then gets all the files in it, and creates local directories accordingly. This client library should provide a callback mechanism for verbosity, e.g. when uploading a file, it should periodically provide the progress of the upload which the client application can show the user.

Data stored in Catalog

All users and VOs has some kind of a distinguished name (DN) which identifies them. These DNs can be used to identify the owner of a Catalog entry. The access control list gives DNs rights such us read or modify.

Each Storage Element has a unique ID, so the list of preferred storage elements is a list of IDs.

Collection

- list of name-GUID pairs
- timestamp of creation
- timestamp of modification
- owner
- access control list
- is closed

File

- checksum
- size
- timestamp of creation
- number of needed replicas
- owner
- access control list
- list of preferred storage elements
- locations (SURLs) with statuses

External

- URL of TSS Gateway
- timestamp of creation
- owner
- access control list

State of a file

The state of a file is given by the state of its replicas:

- unborn: a file has no valid replica, but has a replica which has a state of yet-to-be-uploaded
- improper: a file has at least one valid replica, but not enough
- proper: a file has at least as many valid replica as needed
- deleted: all replicas of the file is marked for deletion

Storage Manager Interface

Each method works on multiple files or collections.

Put

Input: list of (logical name, metadata)
Output: TURLs

Works on files only.
Logical name is the proposed name of the file. Metadata contains size (in bytes), checksum, timestamp of creation, number of needed replicas, owner, access control list, list of preferred storage elements. Size and checksum is manditory, size is needed for scheduling, checksum is needed for replica validity checking. If the list of preferred storage elements is not given, all storage elements accessible for the VO of the given user will be considered.
The Storage Manager first creates an entry in the Catalog if the logical name is not occupied, then the data scheduler modul of the Storage Manager chooses a Storage Element, and tries to initiate file transfer on behalf of the user. The Storage Element gives back an SURL, which can be later used to access this file and a TURL which refers to the initiated file transfer. Then the Storage Manager creates an assertion which allows the user to use the initiated file transfer, then returns the assertion, and the URL of this initiated file transfer (TURL). At last the Maintainer Service will be notified to keep an eye on this file.
After this method there is an entry of this LN in the catalog with one replica (SURL) with a state of yet-to-be-uploaded. If after a certain amount if time the file is still not uploaded it will expire and will be deleted by the Maintainer Service. If a user wants to upload but is not able for some reason (lost the TURL or got error while uploading) it can reinitiate the whole process with Reput.

Reput

Input: list of logical names
Output TURLs

Works on file only.
A user can reinitiate putting the file if it is owned by the user. All existing replica will be deleted and a transfer for a new one is initiated.

Get

Input: list of (logical name [, list of preferred storage elements])

Works on files only.
The Storage Manager gets the locations of the replicas of the file from the Catalog, then chooses one of these SURLs according to the list of the preferred storage elements, then initiates file transfer at the

Storage Element (gets the TURL), creates assertion, and returns them.

Del

Input: list of logical names

Works on files only.
The Storage Manager gets the locations of the replicas of the file from the Catalog, and mark each SURL for deletion. Removes the file from its parent collection, and notify the Maintainer Service to manage the actual deletion of the replicas.

Stat

Input: list of logical names

Gets all the metadata of the file or collection from the Catalog, and returns it. No locations are returned.

Modify

Input: list of (logical name, list of modifications)

Modify metadata of file or collection in the Catalog.

Make

Input: list of (logical name, metadata)

Works on collections only.
The metadata contains the timestamp of creation, the owner, the access control list, the closed-flag, and any arbitrary key-value pairs. This method creates an empty collection, by creating an entry of this collection in the Catalog, if this LN is available, after this collection is created, you can put files in it, or move files to it, then you may close it with Modify.

Unmake

Input: list of logical names

Works on collections only.
Removes the collection itself, but does not remove the file and sub-collections in it.

List

Input: list of (logical name, recursive)
Output: list of (logical name, list of metadata of contents)

If the logical name refers to a file, this returns metadata of that file. If the logical name refers to a collection, then this returns the list of the contents of the collection, with minimal metadata. It can be recursive.

Move

Input: list of (old logical name, new logical name)

If the new logical name is available then creates the entry in the parent collection of the new logical name similar to the entry at the old logical name, then remove the entry of the parent collection of the old logical name. This is a rename, nothing happens with replicas. Returns minimal description with new name or fault with old name.

Link

Input: list of (target logical name, link logical name)

Same as Move, only it does not remove the entry of the parent collection of the target logical name.

Copy

Input: list of (source logical name, target logical name)

Works on files only, recursive copy of collections done by the client library.
It is actually a PUT to the new logical name done by the Storage Manager itself using a replica of the source logical name, or done by the a Storage Element if it is capable.

Glob

Input: list of patterns

Using the Catalog, returns all the logical names matching the given patterns.