Chronopolis Repair Design Document

Michael Ritter, UMIACS October 10, 2016

Background

Within the standard operating of Chronopolis, it is likely due to the volume of data we ingest that we will at some point need to repair data held at a node. In the event a node cannot repair their own data, a process will be in place so that the data can be repaired through the Chronopolis network. In this document a basic design proposal for a protocol through which we can repair collections in a combination of manual and automated work will be outlined.

Considerations

As this design is still living, there are still open questions as to how everything should be finalized and what impact they will have on the final result.

1. What transfer strategy should we use?
    - Node to Node: Transfer between replicating nodes using rsync + ssh with no intermediary step
    - Node to Ingest: Push content to the Ingest node from which a node can repair from
    - ACE: Use ACE with https as the transfer mechanism for serving files
2. Should we create a new client for handling repair, or should it be merged in with the replication service?
    - If it's a new client, what type of application would be best (is cli good enough? do we want a gui? maybe some integration with the ingest server instead?)
3. Should we put a limit on the number of files being repaired in a single request?
4. Should we include tokens in this process, but leave implementation out for now?

Repair Flow

Basic flow: $node_i$ = invalid; $node_v$ = valid

1. $node_i$ sees invalid files in ACE_i
2. $node_i$ gathers invalid files and issues a repair request to the ingest server
    - This can be done in a standalone client

- Might want to consider having multiple requests in the event many files are corrupt
- POST /api/repair

3. $node_v$ sees the repair request
4. $node_v$ checks ACE_v to see if the files are valid
    - If the files are not valid, end this flow here
    - Else: POST /api/repair/<id>/fulfill
5. $node_v$ stages content for $node_i$
    - if through ACE, create a token for $node_i$ and make that available
    - if p2p, make a link (or links) to the files in a home directory for $node_i$
    - if through the ingest server, rsync the files up to the ingest server
6. $node_v$ notifies content is ready for $node_i$
    - POST /api/repair/fulfillment/<id>/ready
7. $node_i$ replicates staged content
    - GET /api/repair/fulfillment?to=$node_i$&status=ready
8. $node_i$ issues an audit of the corrupt files
9. $node_i$ responds with the result of the audit
    - if the audit is not successful a new replication request will need to be made, but we might want to do that by hand
    - POST /api/repair/fulfillment/<id>/complete

Transfer Strategies

Node to Node

Node to Node transfers would require additional setup on our servers, and would likely require a look in to how we deal with security around our data access (transferring ssh keys, ensuring access by nodes is read only, etc). A feasibly staging process could look like:

1. $node_v$ links data (ln -s) in $node_i$'s home directory
2. $node_i$ rsyncs data from $node_v$:/homes/$node_i$/depositor/repair-for-collection

Node to Ingest

Node to Ingest, while lengthy, would have the least amount of development and setup effort associated with it. Since we will most likely not be repairing terabytes of data at a time, one could say this is "good enough". The staging process for data would look similar to:

1. $node_v$ rsyncs data to the ingest server
2. $node_v$ notifies that the data is ready at /path/to/data on the ingest server
3. $node_i$ rsyncs data from the ingest server on /path/to/data

ACE

Repairing through ACE would require additional development on ACE, as it currently does not have any concept of API keys, but otherwise provides the same benefits of Node-to-Node repair with some constraints from http itself. Staging would become quite simple, and amount to:

1. $node_v$ marks the collection as allowing outside access (for API keys only?)
2. $node_v$ requests a new temporary API key from ACE
3. $node_i$ downloads from ACE_v using the generated API key

API Design

The API can be viewed with additional formatting and examples at
`http://adaptci01.umiacs.umd.edu:8080/`

HTTP API

The REST API described follows standard conventions and is split in to two main parts, repair and fulfillment.

Repair API

```
GET /api/repair/requests?<requested=?,collection=?,depositor=?,offers=?>
GET /api/repair/requests/<id>

POST /api/repair/requests
POST /api/repair/requests/<id>/fulfill
```

Fulfillment API

```
GET /api/repair/fulfillments?<to=?,from=?,status=?>
GET /api/repair/fulfillemnts/<id>
```

```
PUT /api/repair/fulfillments/<id>/ready
PUT /api/repair/fulfillemnts/<id>/complete
```

Models

A repair request, sent out by a node who notices they have corrupt files in a collection

Repair Request Model

```
{
    "depositor": "depositor-with-corrupt-collection",
    "collection": "collection-with-corrupt-files",
    "files": ["file_0", "file_1", ..., "file_n"]
}
```

A repair structure, returned by the Ingest server after a repair request is received

Repair Model

```
{
    "id": 1,
    "status": "requested|fulfilling|repaired|failed",
    "requester": "node-with-corrupt-file",
    "depositor": "depositor-with-corrupt-collection",
    "fulfillment": 3,
    "collection": "collection-with-corrupt-files",
    "files": ["file_0", "file_1", ..., "file_n"]
}
```

A fulfillment for a repair, returned by the Ingest server after a node notifies it can fulfill a repair request. Credentials are only visible to the requesting node and administrators.

Fulfillment Model

```
{
    "id": 3,
    "to": "node-with-corrupt-file",
    "from": "node-with-valid-file",
    "status": "staging|ready|complete|failed",
    "credentials": { ... }
    "repair": 1
}
```

Credentials ACE

```
{
        "type": "ace",
        "api-key": "ace-api-key",
        "url": "https://node_v/ace-am" # ?? Not sure if really needed
}
```

Credentials Node-to-Node

```
{
        "type": "node-to-node",
        "url": "node_i@node_v.edu:/homes/node_i/path/to/repair"
}
```

Credentials Node-to-Node

```
{
        "type": "ingest",
        "url": "node_i@chron.ucsd.edu:/path/to/repair"
}
```