APPENDIX E: METAARCHIVE-IRODS TRANSFER

MetaArchive-iRODS Transfer
NHPRC, MetaArchive Project

2012-03-09

Summary
This document describes an automation process that involved the sending of a Folger Shakespeare Library collection (246 archival units measuring 1.3TB) hosted on a University of North Texas (UNT) LOCKSS cache to a storage resource hosted by Chronopolis at the San Diego Supercomputer Center (SDSC).

In order to transfer the content to the iRODS-based Chronopolis data grid, each archival unit needed to be extracted from the LOCKSS cache, and sent via use of the iRODS icommands client utilities.\(^1\) In addition, it was imperative to monitor the transfer at all possible points to detect corruption and make certain that the integrity of the data was maintained. Finally, for the purpose of benchmarking and information gathering, time-based statistics needed to be recorded for each stage of the transfer. These processes were managed by a set of Python-based scripts and utilities that will be described below.

Transfer Requirements
As a preparatory measure, UNT had already solved the issue of extracting archival units from the LOCKSS cache and packaging them into BagIt files using their custom "lockssbagger" utility (see Appendix A: Pre-Transfer Preparation). However, this was only one utility in a larger suite of tools that would be needed to manage a successful automated transfer. Several additional steps needed to be addressed. For each archival unit it was necessary to do the following:

1. Extract the archival unit from the LOCKSS cache and package it into a "filled" BagIt archive.
2. Validate the created bag to make certain that all files were complete.
3. Transfer the created bag to the Chronopolis store via the iRODS client utilities.
4. Obtain a checksum for the transferred files and make certain that they had been transferred without error.

Step 1 would, again, be handled by the custom "lockssbagger" utility. Step 2 would be accomplished through the command line Bag utilities provided by the Library of Congress. Step 3 would employ the "iput" command from the open source iRODS command-line utilities.\(^2\) Step 4 would be accomplished with the "ichksum" command, also from the iRODS utilities.\(^3\) Finally, accounting and clean up would need to be completed upon a successful transfer to the Chronopolis data grid environment.

Automating the Transfer
In order to begin the transfer, a Python script was written with the task of automating and managing the above stages. This script was developed to also maintain a state file in the event of a service interruption, as well as keep a log of the transfer and clean up any unneeded files.

First and foremost, a list of archival units needed to be obtained from the LOCKSS cache. To obtain this list, the script was designed to make an HTTP request to the LOCKSS cache and use

---
\(^1\) icommands: https://www.irods.org/index.php/icommands
\(^2\) iput: https://www.irods.org/index.php/iput
\(^3\) ichksum: https://www.irods.org/index.php/ichksum
the service at the /DaemonStatus?table=Auids URL. This returns an HTML page that contains a table element containing the shortened and full names of all of the AUIDs contained in the cache. In order to get this list into a useable format, the content needed to be parsed out of the table with the “Beautiful Soup” library. For this transfer, in order to select only the AUIDs from the Folger collection, a pattern match was applied to the list of AUIDs, and only the archival units that began with “edu|folger|shakespeare” were kept. This was sufficient, but for future transfer it would be better to have a way to let LOCKSS provide a list of AUIDs for a given collection.

Once this list of AUIDs is parsed, the script can take each AUID and perform the aforementioned steps on each one. In an attempt to better leverage the multiprocessor system that UNT had access to for running the transfer, the decision was made to let the handling of each archival unit be done as an independent process so that UNT could process multiple archival units at once and aim to prevent CPUs from just lying idle while the system was doing network transfers or something similar. This was accomplished with the multiprocessing library, which is now a standard Python package. With this approach, all of the AUIDs would be put into a queue, and a pool of worker processes (three, in our case) would be created to take AUIDs from the queue and process them.

For a transfer such as this, the first worker process would use the custom "lockssbagger" utility to create a bag from a given AUID. It would then call the BagIt tool to make certain that the bag was valid. The second worker process would use the “iput” command to send the bag across to the storage resource assigned for the collection. Once the bag is sent, the third worker process makes use of “ichcksum” to retrieve the remote checksums and compare them to stored checksums in the bag. Finally, once all of this is completed, the third worker process then deletes the local copy of the bag and terminates, so that another beginning worker process can be created to take its place with a new AUID to process.

All of the worker processes remain connected to a message queue in the main process. As they do each step (e.g. bag creation, bag sending, bag validation), they add a message to the queue documenting the current AUID, the time and a status message. These messages are collected by the main process and written to a log file that is saved for future reference. By logging the times and results of each step, it gives us a picture of how long each step takes and may offer clues for future performance enhancement.

**Registering the Transfer**

Once the transfer process reached completion, with each bag successfully delivered, an automated integrity check was required to be run by SDSC. For this process a script, referred to as ‘bagit_manifest_bulk_validation.pl’ was run, which transcends the overall collection directories retrieving the BagIt manifest files and validating stored objects against the manifests.

For each manifest retrieved a checksum subroutine, referred to as ‘sha1sum’ was run against the manifest. The ‘sha1sum’ code reads the manifest, which includes a relative path to the object and the stored sha1 checksum. For each listed object in the manifest, the corresponding stored object is found, and a new checksum is calculated and compared to the authoritative manifest value. If errors are detected they are written to a single logfile. For each BagIt manifest the script creates a validation logfile placing it in the same directory as the manifest. The file is named ‘validation_hostname_dastamp.txt’ and lists all the BagIt objects and their checksum comparison outcomes. The code also does a system ‘ls’ on each BagIt /data directory and compares this to the corresponding authoritative manifest, thereby accounting for any object inventory discrepancies.

Finally to complete the registration of content the number and names of the BagIt files stored at SDSC are checked from an authoritative list generated by the provider, UNT.

---

4 Beautiful Soup: http://www.crummy.com/software/BeautifulSoup/
5 Python Multiprocessing Library: http://docs.python.org/library/multiprocessing.html
Outcomes and Future Development
Using this approach, the transfer of the 1.3TB Folger collection consisting of 246 archival units was a success, and was accomplished in approximately 32 hours. The MetaArchive’s project manager and developers at UNT and Chronopolis were able to watch the logs in real time and monitor the progress of the transfer.

Problems Encountered
All of the content transferred successfully from UNT to SDSC’s storage resource, but an issue arose during the final registration process at SDSC. Due to a mismatch between the manifest files created by the UNT code and the specifications required by the ‘sha1sum’ tool used by SDSC, the final validation and registration process could not fully complete its operations. The formatting problem stemmed from each of the manifest-sha1.txt files having only one space between the checksum and the file pathname, where two spaces were required.

Resolving the formatting issue at the SDSC end required a small customized Python script supplied by the UNT developer. The script simply took the generated manifest-sha1.txt files and re-formatted them with the proper spacing between columns. None of the actual hashes were modified or recalculated. The result was that the ‘sha1sum’ code was then able to successfully validate the manifest files in sequence.

The "lockssbagger" utility was modified to generate manifest files that conformed to the stricter formatting specification in order to ensure a more seamless workflow between the MetaArchive and Chronopolis in the future, and a re-test was scheduled. This re-test provided our project team with the opportunity to test the newly supplied LOCKSS API for its ServeContent servlet. This enables the “lockssbagger” utility to request an authoritative URL and a valid corresponding checksum from a set of separately versioned files that contain non-unique filenames. This is essential for performing a “holey” bag transfer between the MetaArchive and Chronopolis.

The creation of “holey” bags went as expected, with a number of “holey” bags being generated from the Folger collection and being placed on the Chronopolis server to be filled and validated. However, due to scheduling constraints with each of the project partners, there was a period of time between when the bags were created, and when they were validated at Chronopolis. When the time came for the bags to be filled and validated, several errors were generated containing incorrect checksums, and in some cases files that were completely missing or simply not present in the bag manifests. Eventually, this problem was traced to the fact that the transfer had been done on a ‘live’ collection, and that LOCKSS was continually voting, polling and updating the cache. In this manner, the newly versioned files no longer matched those recorded in the “holey” bags.

To avoid disengaging the Folger collection, the decision was made to substitute an archival unit approximating the size of an average Folger archival unit, and which was free from infringing content. Additionally, it was determined that an entire 1.3 TB re-test was not essential to ensure the successful implementation of the Python correction script. The project manager and the development team settled on a subset of multiple copies of the single substitute archival unit now equaling roughly 250 GB.

While running the second set of “holey” bag tests, we did encounter a few issues that needed to be addressed, the first of which being one of network connectivity. The LOCKSS cache was set only to allow access by certain servers and Chronopolis appeared to be accessing the files from a different IP address other than the one specified in the ‘allow’ tables. Pinpointing the problem was difficult, because no sort of IP access logs could be found on the LOCKSS cache server. Eventually, this problem was overcome by simply allowing the range of IP addresses being used at Chronopolis.

The next, more serious, issue was that a number of files in the manifests appeared to be missing
when an attempt was made to fill and verify the bags. Upon investigation, it turned out that, for this given collection, entire directories had been harvested and treated as files, and were given checksums and entries in the file listings from which the manifests were created. This caused problems because, upon 'reconstruction' of the “holey” bags, the directories showed up as just that: directories on the file system. Since these could not be verified and have checksums generated for them, errors arose when comparing the actual files to the list of files in the manifest. This problem was corrected in the “lockssbagger” tool at the point of generating a bag. Simply put, the code would determine if a given entry in the file list was a directory and, if so, would not include a manifest entry for it.

The next issue that we encountered was due to the rather long and unwieldy URLs that were required to fetch the files from the LOCKSS cache through the ServeContent method. The tools utilized by Chronopolis employed the 'wget' tool to retrieve files from the remote server, and when the URLs were passed to wget, it created a conflict due to the way the shell parsed the strings. The solution was to wrap the wget command line strings in quotes, to keep the shell from trying to escape them. This was a change that had to be made in the Chronopolis tool chain.

The last thing to come up was that, even after the other issues had been resolved, Chronopolis occasionally reported the odd failure in file retrieval. These failures could be resolved simply by retrying, but the cause of the failures was never ascertained. Possible culprits could be network issues on either end, or possibly the LOCKSS server becoming burdened down by too many requests. This was not a fatal problem, but further testing would be needed to fully diagnose the issue.

For future development, there are a few things that would likely be worthwhile to address:

- **User Interface:** In its current implementation, the Python transfer manager script, and its use of the custom "lockssbagger" utility, lack a true user interface. Instead of being run on the command line with pre-set operations and flags, it requires that it be loaded in Python’s interactive mode, and supplied with its arguments and settings manually. For a one-time transfer test, this was more than sufficient, but for possible distribution and use by other organizations, a standard user interface would be a necessity.

- **Multiprocessing:** The current multiprocessing model creates workers that take care of all of the operations for a given archival unit. We would likely gain greater granularity of control and more opportunity for resource optimization if we created different worker classes based upon the tasks that they performed (e.g. worker processes to create bags, worker processes to send bags, worker processes to validate sent bags, etc).

**Notes on this document**

This document was drafted by Kurt Nordstrom (UNT), Don Sutton (SDSC), and Matt Schultz (MetaArchive) between 2010-07 and 2011-01, and edited by Matt Schultz on 2012-03.