(Ab)using HPSS
with IU’s Media Digitization and Preservation Initiative

Brian Wheeler (bdwheele@indiana.edu)
Indiana University Libraries

HUF 2019
Indiana University
October 17, 2019
Quick disclaimers

• I’m an HPSS end-user
  • Some details are most likely wrong
• I actually like tape
  • This makes me an outcast in the archive/preservation world
• The IU HPSS crew are pretty awesome
  • I tend to make their lives difficult, but they’ve not banned me….yet.
Media Digitization and Preservation Initiative

“… to digitize, preserve, and make universally available by IU’s bicentennial [2020] – subject to copyright or other legal restrictions – all of the time-based media objects on all campuses of IU judged important by experts” -- IU President McRobbie, October 2013

- Goal of 325,000 A/V objects and 25,000 Film objects
- Estimating >20PB of content when finished
- [https://mdpi.iu.edu](https://mdpi.iu.edu)
Digitization Sources

- Two vendors digitizing content:
  - Memnon Archiving Services (a division of Sony)
    - Bulk digitizing using a factory model
    - Formats which are “easy”: VHS, Audiocassette, Open Reel, Betacam, etc.
    - Also digitizing Film
  - IU Digitization Studios
    - Boutique model of digitization
    - Harder formats: Wax cylinder, lacquer discs, Betamax, etc.
    - Also any objects which are difficult for Memnon to do (multi-speed tape)
Ingest Rates

• For A/V the initial estimate was 8-10T per day
  • Hundreds of objects, very different sizes
• Film estimate was an additional 25T per day
  • Fewer objects, but each substantially larger
• Every object requires
  • Validation
  • Derivative creation
  • Metadata collection
Original Data flow design

- IU
- Memnon
- Working Disk
- Processing
- Derivatives
- Avalon
- The Internet
- QC Stations
- HPS S
HPSS as the working storage

• Money wasn’t allocated for processing hardware or working storage
• Non-technical decision to use HPSS as the working storage
  • HPSS already has a bunch of disk for caches
  • All disk is the same, right?
  • Let’s use that

• My reaction to the decision:
  • “I’m excited to work with you on this”
MDPI Objects
What is an object?

- Representation of a physical object: CD, LP, film reel, wax cylinder, etc.
- Contains a manifest with vendor-supplied checksums
- Stored as a directory structure
  - Structure is compliant with Library of Congress BagIt 0.97
  - Files from vendor are separated from generated (derivative) files
  - Mix of file size from bytes to terabytes
- Objects are never deleted
  - All object directories are timestamped and replacements are tracked in a database
  - Generated files may be replaced within an object
Object Sizes

- Size is entirely dependent on length and type:
  - Audio is 4GB/hour
  - Video is 64GB/hour
  - 2K Film is 1.5TB/hour
  - 4K Film is 6.0TB/hour
- Some objects have more than one sequence (side)
  - File count increase is (nearly) linear
  - Object size is determined by the total run time of all sequences
<table>
<thead>
<tr>
<th>Type</th>
<th>Count</th>
<th>Avg Size(G)</th>
<th>Max Size(G)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-Inch Open Reel Video Tape</td>
<td>4,082</td>
<td>36.33</td>
<td>149.77</td>
</tr>
<tr>
<td>1/2-Inch Open Reel Video Tape</td>
<td>398</td>
<td>26.70</td>
<td>64.33</td>
</tr>
<tr>
<td>45</td>
<td>4,161</td>
<td>0.52</td>
<td>1.40</td>
</tr>
<tr>
<td>78</td>
<td>37,286</td>
<td>0.47</td>
<td>1.70</td>
</tr>
<tr>
<td>8mm Video</td>
<td>1,041</td>
<td>77.88</td>
<td>162.98</td>
</tr>
<tr>
<td>Aluminum Disc</td>
<td>1,308</td>
<td>0.66</td>
<td>1.78</td>
</tr>
<tr>
<td>Audiocassette</td>
<td>53,231</td>
<td>4.82</td>
<td>12.72</td>
</tr>
<tr>
<td>Betacam</td>
<td>19,353</td>
<td>43.62</td>
<td>377.99</td>
</tr>
<tr>
<td>Betamax</td>
<td>1,266</td>
<td>84.96</td>
<td>289.20</td>
</tr>
<tr>
<td>CD-R</td>
<td>11,966</td>
<td>1.19</td>
<td>2.86</td>
</tr>
<tr>
<td>Cylinder</td>
<td>6,734</td>
<td>0.31</td>
<td>1.30</td>
</tr>
<tr>
<td>DAT</td>
<td>9,015</td>
<td>2.77</td>
<td>4.38</td>
</tr>
<tr>
<td>DV</td>
<td>17</td>
<td>20.07</td>
<td>37.49</td>
</tr>
<tr>
<td>DVD</td>
<td>2,982</td>
<td>6.93</td>
<td>55.03</td>
</tr>
<tr>
<td>Film</td>
<td>17,307</td>
<td>378.16</td>
<td>4016.39</td>
</tr>
<tr>
<td>Lacquer Disc</td>
<td>6,132</td>
<td>1.08</td>
<td>49.08</td>
</tr>
<tr>
<td>LP</td>
<td>38,841</td>
<td>2.96</td>
<td>6.15</td>
</tr>
<tr>
<td>Open Reel Audio Tape</td>
<td>68,814</td>
<td>2.10</td>
<td>25.75</td>
</tr>
<tr>
<td>Other Analog Sound Disc</td>
<td>18</td>
<td>0.93</td>
<td>1.00</td>
</tr>
<tr>
<td>U-matic</td>
<td>15,344</td>
<td>35.50</td>
<td>130.53</td>
</tr>
<tr>
<td>VHS</td>
<td>46,139</td>
<td>107.62</td>
<td>857.29</td>
</tr>
</tbody>
</table>
Example object

MDPI_40000001816000/
  bag-info.txt
  bagit.txt
  data/
    MDPI_40000001816000.xml
    MDPI_40000001816000_01_access.mp4
    MDPI_40000001816000_01_pres.wav
    MDPI_40000001816000_01_prod.wav
  generated/
    MDPI_40000001816000_01_access_ffprobe.xml
    MDPI_40000001816000_01_high.mp4
    MDPI_40000001816000_01_high_ffprobe.xml
    MDPI_40000001816000_01_med.mp4
    MDPI_40000001816000_01_med_ffprobe.xml
    MDPI_40000001816000_01_pres_ffprobe.xml
    MDPI_40000001816000_01_prod_ffprobe.xml
    MDPI_40000001816000_group.txt
    MDPI_40000001816000_mods.xml
    MDPI_40000001816000_pod.xml
    README.txt
  manifest-md5.txt
  tagmanifest-md5.txt
Processing Environment
Real hardware is AWESOME!

- Shared VMs are great for lots of things
  - Transcoding is not one of them
- Dedicated hardware is great for transcoding
  - Lots of simultaneous transcodes
  - Easier tuning since I’m the only tenant!
  - Spare CPU cycles used for cleanup work
- Hardware quite stable over 5 years
  - 2-3 replaced disks
  - 1 memory error requiring a DIMM reseat
- 10GbE Interconnects

Not awesome.
Hardware by the numbers

- QC-01: Working QC storage for A/V
  - Lenovo x3650m4, 9T spinning disk
- Xcode-01 – Xcode-04: A/V Transcoders
  - Lenovo x3650m4, 48 threads, 128G, 1.4T scratch disk
- QC-02: Working QC storage for Film + Film Transcode
  - Dell r730, 72 threads, 256G, 7.5T scratch SSD, 120T QC disk
- Xcode-05 – Xcode-07: Film Transcoders
  - Dell r730, 72 threads, 256G, 7.5T scratch SSD
- VC-01, VC-02: Vidicert QC Servers
  - Running proprietary QC tools on Windows Server
Software Environment

- RedHat Enterprise Linux 7 on x86_64
- Cron-driven processing
  - Start scripts every minute, if not already running
  - Reports issues via email automatically
  - Individual scripts can be paused or run by hand
- Workflow software in Perl
  - Familiar with language, has most needed libraries
  - 6 month from start of coding to production
  - Next project -- Python
Lots of “simple” tools

- Eight scripts control the entirety of the workflow
  - Most workflow scripts < 200 LOC
  - Manual processes
    - ~100 Maintenance / reporting scripts
    - Changing state, pausing workflows, generating reports, etc
- Current Stats:
  - 10K LOC for support libraries
  - 400 LOC on average
  - 18K LOC for maintenance scripts
    - Most are less than 100 LOC
The object state machine
Tape motion part of the workflow
Interfacing HSI
Pre-MDPI HSI interfaces

• Originally the Library used shell scripts to push data via HSI
• In 2007 I wrote a Java wrapper for HSI
  • One method = one hsi command
  • hsi binary started for each command, output parsed
  • Background stage commands are useless, as they’re cancelled on client exit
  • Terrible latency on methods
MDPI HSI interface

- Written in Perl
- Methods for most commands
  - Commands piped to underlying process, output parsed (including errors)
- Per-PID hsi sub-process
  - hsi process started on first call
  - New hsi started if the pid has changed (i.e. after a fork)
  - Process continues receiving commands until script exits
- Multiple commands separated by ‘;id’
  - Id output acts as a sentinel to indicate end of command
MDPI HSI interface (2)

- Reasonably low latency on calls
  - Still hundreds of milliseconds
- Filenames are normalized to keep accesses within a root
- On disconnect, it will try to reconnect
  - Some commands are retried
- Adding new commands is pretty easy
  - Parsing output is a bit hard sometimes
- A Python version now exists for future tools at the library.
HSI example code

my $hsi = $mdpi->getHSI();
if($hsi->exists($manifest) && !$hsi->needsStaged($manifest)) {
    if($hsi->get($manifest, ".")) {
        print "Retrieved $manifest\n";
    } else {
        print "Failed to retrieve $manifest\n";
    }
} else {
    print "$manifest doesn’t exist or isn’t on the disk cache\n";
}
Further development

- Python rewrite for the HSI library
  - Fixes some bugs, and removes some Perl-isms
- Simple HTTP download proxy using the Python version
  - Path + token authentication
  - Returns content if the matching path is in the disk cache
  - Returns “try again later” if it is not currently in the cache
    - Initiates a stage and will return “try again later” until it is available
  - HEAD method returns information about the path
    - A directory list, or stat()-style information otherwise
Data Flow Walkthrough
Initial Ingest

- IU runs one shift per day
- Memnon runs two shifts per day and the number of streams varies depending on the configuration of the studios
Object ownership

- Content from Memnon is owned by their account
  - In the end, the library needs to own the content
- Jeff Russ wrote a tool that would do a restricted chown
  - My code SSH’s to an HPSS node and runs the suid code with the file paths
  - Usually very fast…but it will hang if migration is in progress
- Once we own them, they’re made private
  - Group and World read/write removed
Data integrity

• This is a preservation project
  • Checksums in manifest must be verified against actual content
  • Content must be valid from start to end of processing
  • Content on tape must be exactly the same as what was sent
• Integrity problems from the past:
  • Network router RAM with stuck bit prior to TCP checksum generation
  • Memory corruption on server, creating bad cache data (but data stored OK)
Verifying content

- HSI’s in-flight checksum calculation is incredibly helpful
  - Generated checksum can be compared against the manifest
  - And verify the content made it to the different machines correctly
- What about the tapes?
  - HPSS version we were running didn’t support E2EDI
  - No guarantee that the tape copy matched the stored checksums
Use hashverify to verify tapes?

- At first glance, seems like the perfect solution
- Has some undesirable features:
  - Wants to read the cached version by default (not the tape)
  - Content is sent to the client for checksum computation and discarded
- Given the size of our content, hashverify will not work
  - Another full transfer of the content wasn’t feasible

- Only option is to re-read the tapes
Re-reading tapes introduces problems

- Data must be purged from the disk cache
  - Purging can hang until the migration finishes
- Staging data takes time
  - Individual objects may reside on multiple tapes
  - Staging isn’t as optimal as one would hope
- Tape contention is an issue
  - New data is being written to the same tapes we need to re-read
Bad object shortcut

• No reason process to process malformed objects
  • Manifest checksums compared against HSI checksums
  • Manifest checked for missing or extra files
  • Object checked to make sure the expected files for the object type are present
• Any deviation rejects the object
  • No need to process any further, because it is not well-formed
• All of these are metadata checks
Tape contention

• Initially a single COS was created to store all of the MDPI data
  • Keeps the MDPI data separate from the general population data
  • Helps keep track of who’s buying the tapes
• Using multiple COS solves part of the tape contention issue:
  • A ‘master’ COS was added for content from the digitizers
    • Written by digitizers, read by transcoders
    • Film and A/V have separate master COS
  • A ‘derivative’ COS was added for content created by the transcoders
    • Written by the transcoders, read by the distribution system
    • All formats share the same COS
Object data flow

- Digitizer writes object to disk cache

- HPSS migrates object to master COS tapes (both sites)

- Object in disk cache is purged
Object data flow (2)

- Object is staged to the cache

- Transcoder downloads object for processing (verifying checksums)

- Transcoder transfers newly-created derivatives to HPSS
Object data flow (3)

- HPSS migrates object derivatives to derivatives COS (async)

- Distribution process downloads derivatives (usually 30-40 days later)
Multiple object contention

- Many objects are in flight at once
  - Don’t stage an object while another is being migrated on the same tape
  - No control over when objects are migrated
  - No information on which tape will get used next for new data

- Only way is to wait for the system to become idle
Scheduling purging and staging

- Digitizers have two six-hour windows to transfer content
  - Entire window is usually not needed
- As objects are sent:
  - Wait for all of the object files to be migrated (migrate_wait)
  - Issue the purge command for the files (purging)
  - When the purge is complete, mark the object as ‘purged’.
- When the digitizer has finished and everything is in ‘purged’ stage
  - Issue a stage for all of the purged objects (staging)
  - Poll for completion (accepted)
Tape motion part of the workflow
Special cases

• Purge cycle bypass
  • Initially film content was taking too long to do 100% purge/stage
  • 50% content was re-read from tape
  • Tuning solved it, and we’re back at 100%

• Delayed purge
  • Handles when the object is in the process of migrating and hangs the purge
Actual data flow diagram
Staging
Staging efficiency

• The purge/stage cycle is a huge bottleneck in processing
  • I want it to be as fast as possible
  • But I have to be a good citizen on this shared resource
• Built-in staging in hsi (and probably HPSS)
  • Doesn’t seem to work as well as it could
  • Isn’t really tuned for the kind of workflow I’m trying to do
• Could I do better?
Problems with staging from hsi

• Background stages…sort of work
  • On 2nd background stage, hsi waits for first to complete
• Lots of contention when many clients are staging on the same tapes
  • Requests are serviced as a FIFO
  • No awareness that the tape may be positioned optimally for another request
  • Tapes may be dismounted only to be remounted a few minutes later

• Luckily, hsi provides enough information to optimize staging
MDPI Staging Server

• MDPI scripts can request files for staging
  • All file requests put into a common pool
  • Scripts can wait for staging request to finish
    • OR they can poll the files and process them as they are staged
• Staging server
  • Groups and sorts all outstanding requests from all MDPI scripts
  • Issues stages by tape, in section and offset order
  • Can be restarted without losing outstanding requests
Staging Requests

- Requests consist of
  - A priority (0 = lowest, 1 = normal, 2 = priority)
  - A wait flag: library will block until request is fulfilled
    - If not waiting, application can call `$hsi->isOnDisk($filename)` to poll
  - A list of files to stage
- Scripts don’t need to know anything beyond filenames
  - All of the gross details are handled by the server
Parsing requests

- Tape, section, and offset are queried for each file
  - This is time consuming, but it is multi-threaded
- Requests are split into different lists, grouped by tape
  - These lists will be merged into staging requests later
- In the end we have a pile of lists for each tape, each having:
  - Priority
  - # of files to stage
  - Time of request
Selecting the next tape to stage

- Outstanding tapes are sorted by highest priority, number of files, and date.
- The first one on the list is the next tape to stage:
  - Unless we’re already staging 12 tapes, then we wait for completion.
  - If it is already in use…then go to the next one in the list.
- Building the staging request:
  - All of the request lists are merged and sorted by tape section and offset.
  - Stage command is run for each file in the list, in order.
  - Files within an aggregate are all staged with a single hsi command (up to line limit).
## Status page

### Current

<table>
<thead>
<tr>
<th>Queued</th>
<th>Active</th>
<th>Requests</th>
<th>Files</th>
<th>Tapes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
<td>5</td>
<td>201</td>
<td>5</td>
</tr>
</tbody>
</table>

### Active Tapes

<table>
<thead>
<tr>
<th>Tape</th>
<th>Seg-Off</th>
<th>Start Time</th>
<th>QPos</th>
<th>COS</th>
<th>Filename</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1052700</td>
<td>158+0</td>
<td>2019-09-20-150525 (18s)</td>
<td>24 / 29</td>
<td>9</td>
<td>MDPI_40000003810845_01_mezz.mp4</td>
</tr>
<tr>
<td>D1053800</td>
<td>84+0</td>
<td>2019-09-20-150522 (21s)</td>
<td>11 / 15</td>
<td>9</td>
<td>MDPI_40000003810894_01_mezz.mp4</td>
</tr>
<tr>
<td>D1053700</td>
<td>845+0</td>
<td>2019-09-20-150541 (28s)</td>
<td>14 / 17</td>
<td>9</td>
<td>MDPI_40000003810894_01_pres.iso</td>
</tr>
</tbody>
</table>

| Completed | 46 / 61 |

---
Another view of status page

Current

<table>
<thead>
<tr>
<th>Queued</th>
<th>Active</th>
<th>Requests</th>
<th>Files</th>
<th>Tapes</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>12</td>
<td>18</td>
<td>15431</td>
<td>18</td>
</tr>
</tbody>
</table>

Active Tapes

<table>
<thead>
<tr>
<th>Tape</th>
<th>Seg+Off</th>
<th>Start Time</th>
<th>QPos</th>
<th>COS</th>
<th>Filename</th>
</tr>
</thead>
<tbody>
<tr>
<td>D0920100</td>
<td>28+699701590</td>
<td>20190916-165303</td>
<td>174s</td>
<td>270 / 2647</td>
<td>MDPI_40000003248301_01_mezzRaw_VidiCert.tar (aggregate head 15)</td>
</tr>
<tr>
<td>D0920800</td>
<td>1115+1287992224</td>
<td>20190916-165519</td>
<td>38s</td>
<td>155 / 1279</td>
<td>MDPI_40000003421262_01_mezzCrop_VidiCert.tar (aggregate head 2)</td>
</tr>
<tr>
<td>D0921100</td>
<td>900+5888339261</td>
<td>20190916-164515</td>
<td>642s</td>
<td>15 / 167</td>
<td>MDPI_40000003363670_01_mezzRaw_VidiCert.tar (aggregate head 54)</td>
</tr>
<tr>
<td>D0921300</td>
<td>550+54485255756</td>
<td>20190916-165313</td>
<td>164s</td>
<td>87 / 2410</td>
<td>MDPI_40000003372598_01_mezzCrop_VidiCert.tar (aggregate head 4)</td>
</tr>
</tbody>
</table>
Further development...

- Python-based library for stage management
  - Manages a database of outstanding requests
  - Client can either
    - Talk directly to back-end database via API
    - Use a REST API to manage the database
  - All of the gory details are hidden from the client
- No fancy web status page
  - Would be easy to write, though
Odds and Ends
MDPI Scheduler

- Every node runs a scheduler script every minute via cron
  - It decides what scripts should run on this particular node
  - Any (or all) scripts can be put on hold for debugging
  - When in ‘panic’ mode, it will immediately exit so no processing happens
- Scheduler will not run if HPSS is unavailable
  - An hsi ping (a connect and run ‘id’) determines if HPSS is up
  - If it isn’t, an email is sent, and it will continue to retry until available
  - When available, it will send an email listing the start and end times
Scalability Issues

• hsi commands have a relatively large latency
• Scripts may have hundreds of paths to query
• Solution:
  • Most scripts are heavily multi-processing
  • Not uncommon to have 10-15 hsi processes per script
  • Admins had to increase the number of concurrent logins
Wait vs No Wait when staging

- Early scripts were written to submit the stage for all objects and wait
  - Script sat idle while waiting for tape drives to do their thing
- Newer scripts tend to try to overlap waiting and working
  - Stage list is submitted with ‘no wait’ flag
  - Script walks through the staged files and queries HPSS if the file is on disk
    - If it is, start processing it immediately, and remove it from the todo list
    - If no files were staged during this pass, wait 30 seconds and scan again
Directory Limits

- Original MDPI design kept all active objects in a single directory
- As each object is itself a directory, we hit the limit of ~32,000 directories
- Had to hash objects into subdirectories, based on last 3 digits of barcode
  - No directory has more than 1000 entries
  - Evenly distributed across all directories
Content lockdown

• Fear that the user account could be compromised and data removed
• Created a virtual copy of the data:
  • Identical directory structure
  • Files are hard linked to original data
    • If working copy is deleted, the link count remains > zero
• Top level of the copy has ownership set to HPSS admins
  • I can’t see or modify the content, so data cannot be deleted by my account
Offline backup

- 3rd copy of tapes made for offline usage
  - A/V master tapes have been copied
  - Film master tapes are starting soon
    - Content data issue has slowed this down
  - Derivative tapes not started
    - Due to number of very small files, a single tape took several weeks
    - Looking for a new plan
- Will need to do it again, to catch everything created after we started
Repairing corrupt data

- Vendor sent tarballs of BagIt bags for films
  - Bags generated by an in-house library
  - Used “Bagit-version” rather than the proper “Bagit-Version”
  - 2PB of files need to be repaired:
    - Download tarball
    - Patch ‘v’ -> ‘V’
    - Patch manifest checksum
    - Upload new tarball to HPSS
HPSS for content delivery?

- Lots of derivative streaming files to serve to end users
  - Lots of disk space
  - Small percentage of all files will likely be viewed
  - As before, storage money was not allocated
- Admin: “Can’t we just serve this content directly from tapes?”
  - This time, it was “NO!”
  - Derivatives stored on a local S3 device
HPSS/HSI Wishlist
A programmatic interface would be great

- Sending commands & parsing human-readable output is gross
  - Using hsi with the commands and output as JSON would be an improvement
- Multiple language support
  - Python and Java are probably my two most wanted
- S3-style interface would be OK
  - But there should be a way to get more info for staging, etc
  - File lists are notoriously slow for S3
Data integrity

- Store not just the checksum of the data, but when it was last computed
- Ability to initiate media-level validation
  - And get a list of files which are on the just-validated media
- Ability to run fixity check on a file using a nonce to seed the algorithm
  - Should satisfy even the most paranoid collection manager!
Project Statistics
By the numbers

- 52 months of operation – 0 hours of unplanned downtime!
- 20 different physical formats
- ~334,000 objects, 13,000 film, 321,000 A/V
- 290,000 hours of content
- 12PB of active content, 13PB total tape storage
- 8 months with daily average ingest > 20TB/day
- 30 days with ingest > 35TB, max ingest day is 39.95TB
Data Ingest Rates

Average Ingest Per Day (TB)
Where we stand now...

MDPI Data Size and Hours of Content

- Audio Size
- Video Size
- Film Size
- Total Size
- Audio Duration
- Video Duration
- Film Duration
- Total Duration
Questions?

For more information:
https://mdpi.iu.edu
https://avalonmediasystem.org

Brian Wheeler
bdwheele@Indiana.edu