




Large Scale Digital Repositories executable workflows for large-scale execution

Authors

William Palmer (British Library), Bolette Ammitzbøll Jurik, Rune Bruun Ferneke-Nielsen (State & University Library Denmark), Opher Kutner (ExLibris), Sven Schlarb (Austrian National Library), Clemens Neudecker (National Library of the Netherlands), Matthias Hahn (Fachinformationszentrum Karlsruhe)

May 2014

This work was partially supported by the SCAPE Project. The SCAPE project is co-funded by the European Union under FP7 ICT-2009.4.1 (Grant Agreement number 270137).

This work is licensed under a CC-BY-SA International License 

Executive Summary

In the SCAPE project, the role of the testbeds is to employ SCAPE platform technology as well as preservation components to develop workflows that can be used to process very large data sets.

The Large Scale Digital Repository Testbed develops workflows for processing data that does not fit within the web content or research content testbeds. At a high level the workflows developed within this testbed cover format migration, format characterisation, identification of preservation risks in content, quality assurance, large scale ingest and repository profiling.

Since the previous deliverable D16.1¹, the testbeds as a whole have refined the preservation scenarios and requirements, and provided input for the development of new tools and the adaption of existing software. By using large data sets from a real-world production context, the testbeds were able to provide feedback, bug reports and define further requirements for component developers.

In this deliverable, the large-scale workflows are described in the context of the different user stories with a focus on the technical solutions. Datasets and details about the data is only mentioned if this is required in order to explain technical decisions that have influenced workflow design.

Detailed evaluation results are not reported here, unless they informed design choices. Full results will be reported in deliverable D18.2 - "SCAPE final evaluation and methodology report".

¹ <http://www.scape-project.eu/deliverable/d16-1-lsdr-executable-workflows-for-experimental-execution>

Table of Contents

Deliverable	i
Executive Summaryiii
1. Introduction.....	5
2. Refining Scenarios to User Stories, Experiments and Evaluations.....	5
3. Following on from D16.1.....	6
4. Large Scale Digital Repository Testbed User Stories.....	6
4.1. User Story: Characterisation of Large Audio and Video Files.....	6
4.1.1. Experiment: Characterisation and validation of audio and video files during ingest	7
4.2. User Story: Large Scale Audio Migration.....	9
4.2.1. Experiment: SB Experiment Audio MP3 to WAV Migration and QA on Hadoop Cluster.....	9
4.3. User Story: Large Scale Image Migration	12
4.3.1. Experiment: KB Metamorfoze Image Migration & QA.....	12
4.3.2. Experiment: BL Newspapers on the BL Platform	15
4.4. User Story: Large Scale Ingest	22
4.4.1. Experiment: Large Scale Ingest with Fedora4	22
4.5. User Story: Policy-Driven Identification of Preservation Risks in Electronic Documents	26
4.5.1. Experiment: Validate PDF&EPUBs and check for DRM	26
4.6. User Story: Quality Assurance of Digitized Books	27
4.6.1. Experiment: Quality Assurance of Digitized Books Experiment.....	27
4.7. User Story: Repository Profiling	34
4.8. User Story: Validation of Archival Content Against an Institutional Policy.....	35
4.8.1. Experiment: Validate JPEG2000 Newspaper scans Using Jpylyzer	36
5. Other Large Scale Execution Methods	37
5.1. Using Rosetta.....	37
5.2. Using Microsoft Azure	40
5.3. Using Apache Pig within the Execution Platform	40
6. Conclusion and next steps.....	40
7. Glossary	41

1. Introduction

This deliverable describes the preservation workflows that have been developed by the Large Scale Digital Repository Testbed (LSDRT) for execution on the SCAPE Platform.

The real-world large scale datasets that are in use by LSDRT are typical of the content that is held by partner organisations that is not covered by the Web Content or Research Data testbeds:

- Audio files
- Video files
- Image files
- Document/eBook files
- Metadata

The types of preservation workflow that have been developed are also quite varied and at a high level include:

- Characterisation of content
- Migration of one file format to another
- Validation that content matches an institutional policy
- Identification of files that contain preservation risks
- Large scale ingest of metadata

There are a wide variety of different workflow types that are covered within this testbed package. The workflows make use of tools developed within the PC subproject (PC.WP.1 Characterisation Components, PC.WP.2 Action Services, and PC.WP.3 Quality Assurance) and in some cases the developers of those tools are active within the LSDRT work package.

In the two years since the previous deliverable, D16.1, several large changes within the work package, including staff turnover, and the change from “Scenarios” to “User Stories” occurred. Also, the initial large scale testing of the experiments in the User Stories was completed. The full results from the large scale testing of the workflows described in this deliverable will be reported in the D18.2 deliverable, in project month 44.

The main platform that is in use is the SCAPE Platform, but we also have some testing performed on the Rosetta platform, a commercial digital preservation system. Extensive large scale testing on the Rosetta platform has not been possible due to no partner institution having a local Rosetta installation. Work has also been completed on web services for office format characterisation and migration using the Microsoft Azure platform.

The User Stories for this testbed are described in the next section, followed by sections discussing Rosetta, Azure, Apache Pig, a section detailing the next steps for this work package and finally, the conclusions.

2. Refining Scenarios to User Stories, Experiments and Evaluations

The Scenarios, as described in D16.1, had become large and contained so much information that the actual problem the scenario was trying to address wasn't as clear as it could be.

After some consideration the Scenarios were refined into a streamlined, agile approach consisting of:

1. A **User Story** - a short and succinct high-level statement of a preservation issue
2. Each User Story can have one or more **Experiments** - an implementation of a solution for the User Story - at this level particular details are documented. An Experiment includes details of the dataset, preservation components, workflow and processing platform type. Some User Stories have more than one experiment, where there are different organisations working on the same User Story.
3. Each Experiment can have one or more **Evaluations** - details of a particular execution of an Experiment, which includes details of the runtime environment, metrics according to the Metrics Catalogue² and any other information pertinent to that particular execution.

3. Following on from D16.1

Work has been completed on the next steps that were identified within D16.1. Microsoft Azure web services for migration of office formats are available, as well as a new User Story for detecting DRM within PDF and EPUB files.

User Stories and Experiments have been developed that address the needs of organisations, and they have been demonstrated to scale to large-scale datasets. The Experiments are using software that has been developed elsewhere within the project and good links have been fostered with other work packages.

In D16.1 we planned to use the Results Evaluation Framework (REF) to evaluate the results of testbed results. The sub-project has since established an internal evaluation methodology which no longer requires the REF. The results from the large-scale execution of workflows are captured in the Evaluations for work package 18 (Evaluation of Results) using the relevant metrics as defined by that work package.

4. Large Scale Digital Repository Testbed User Stories

The Experiments described below have all been tested at scale (1TB or greater), whenever possible. In the next section we will describe the high level user story, and then detail each experiment.

We are aware of the variation of detail of information within this and the following section. This is a reflection of both the technical and organisational decisions and environment used within the experiments. We felt it was important to capture this information within this deliverable.

4.1. User Story: Characterisation of Large Audio and Video Files

This user story is:

² <http://wiki.opf-labs.org/display/SP/Metrics+catalogue>

As the owner of a large collection of video files I need a digital preservation system that can characterise very large audio/video files to enable me (or a preservation watch system) to evaluate the collection for preservation risks and perform ongoing risk management.

The user requirements are:

1. We need to be able to characterise very large (8GB+) video files
 1. This includes identifying container formats and contained streams
 2. Features extracted need to be decided - we need to consider what is useful to extract and include here as requirements. See:
 1. JISC Digital Media infokit: High Level Digitisation for Audiovisual Resources³
 2. JISC Digital Media Guide: Metadata and Digital Video⁴
 3. Mike Coyne and Mike Stapleton: “The Significant Properties of Moving Images” (JISC Digital Preservation Programme: Study on the significant properties of Moving Images), March 2008⁵
2. We need to perform characterisation quickly and efficiently
3. It would be good to be able to validate video format compliance with specification

Developers note that some existing characterisation tools (JHOVE2 for instance) do not seem to work well on large files. For JHOVE2 this has been submitted as a bug⁶. Also note this story provides the opportunity to compare tools over time, such as the Tika/DROID/etc. This has not been put into an experiment on audio and video files due to sparse resources.

4.1.1. Experiment: Characterisation and validation of audio and video files during ingest

Dataset	Danish TV broadcasts, mpeg videos and mpeg-2 transport stream and Danish Radio broadcasts MPEG1-Layer 2.
Platform	SB Video File Ingest Platform ⁷
Workflow(s)	Taverna workflow part of the Danish State and University Library (SB) “youseeingestworkflow” Github repository ⁸ .

This characterisation is performed at ingest time, when new data is added to the Danish State and University Library Radio/TV collection. The daily Radio/TV broadcast ingest is almost 1TB.

We describe an in-production Taverna audio and video ingest workflow with both characterisation and audio and video format validation. The tool used here is FFprobe⁹. We considered an experiment

³ <http://www.jiscdigitalmedia.ac.uk/infokit/audiovisual-digitisation>

⁴ <http://www.jiscdigitalmedia.ac.uk/guide/metadata-and-digital-video/show-hidden>

⁵ http://www.jisc.ac.uk/media/documents/programmes/preservation/spmovimages_report.pdf

⁶ <https://bitbucket.org/jhove2/main/issue/181/do-not-fail-on-2gb-files-when-having-2gb>

⁷ <http://wiki.opf-labs.org/display/SP/SB+Video+File+Ingest+Platform>

⁸ <https://github.com/statsbiblioteket/youseeingestworkflow>

⁹ <http://www.ffmpeg.org/ffprobe.html>

using FFprobe across all repository content. However, this tool is too fast to give any scale problems, as it only reads the header of the files.

The Taverna audio and video ingest workflow below uses the FFprobe tool for characterisation. The characterisations are compared with the specification for the audio and video files to be ingested using Schematron¹⁰. The specifications were translated from human readable format to Schematron by hand. We note that while FFprobe + Schematron is not satisfactory as a file format validation tool it can reveal files with audio and video formats not complying with the specification.

The Taverna workflow is shown in the next figure.

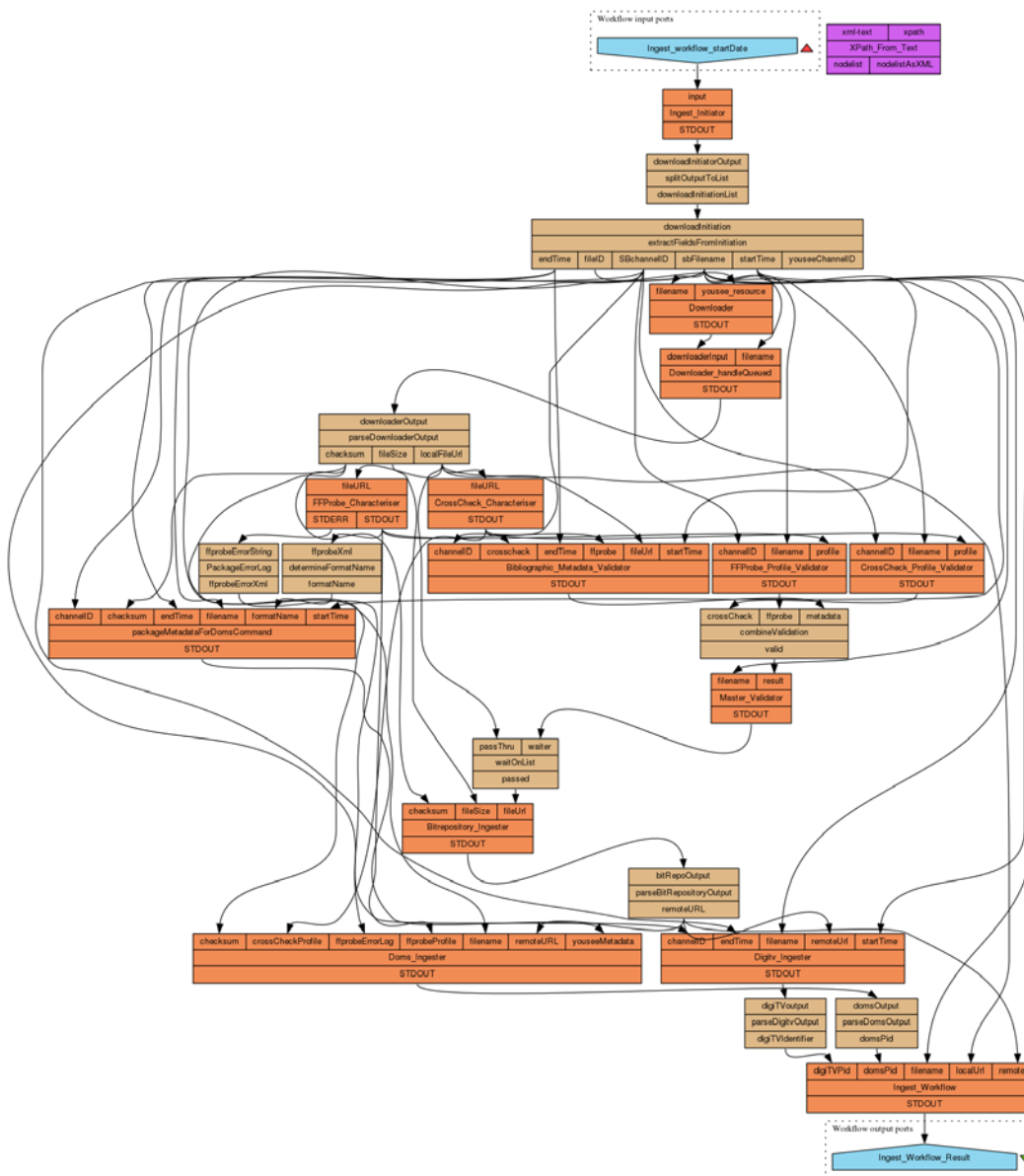


Figure 4.1.1-1: Taverna workflow for large scale characterisation and validation

¹⁰ <http://www.schematron.com/>

This ingest workflow is responsible for retrieving the requested audio and video broadcasts list and downloading the corresponding files and associated metadata. The files are then characterised using FFprobe, and based on these characterisations, the audio and video file formats are validated against the specification using Schematron¹¹. If they validate, the files are then saved in the SB Bit Repository¹² and the metadata is saved in DOMS¹³, the SB metadata repository.

4.2. User Story: Large Scale Audio Migration

The user story is:

As the owner of a large audio collection, I need a digital preservation system that can migrate large numbers of audio files from one format to another and ensure that the migration is a valid and complete copy of the original.

User Requirements/Components:

1. We need to be able to migrate MP3 to WAV
2. We need a measure of similarity between two audio files based on how they 'sound'
3. We need to be able to compare the properties of MP3 files with the corresponding properties of the migrated WAV files

4.2.1. Experiment: Audio MP3 to WAV Migration and QA on Hadoop Cluster

Dataset	Danish Radio broadcasts, MP3 ¹⁴
Platform	SB Hadoop Platform ¹⁵
Workflow(s)	Workflow Entry: "Slim Migrate And QA MP3 to WAV Using Hadoop Jobs" on MyExperiment ¹⁶

Last year an experiment was undertaken at SB using the SCAPE tool suite xcorrSound¹⁷ and a Taverna workflow. This experiment was reported on in the SCAPE Deliverable *D16.1 LSDR Executable Workflows for Experimental Execution*¹⁸ and it is also documented on the SCAPE wiki¹⁹

This workflow contained

- Migration from MP3 to WAV using FFmpeg
- Extracting and comparing properties of the original and the migrated files using FFprobe

¹¹ <http://www.schematron.com/>

¹² <https://sbforge.org/display/BITMAG/The+Bit+Repository+project>

¹³ <https://sbforge.org/display/DOMS/Home>

¹⁴ <http://wiki.opf-labs.org/display/SP/Danish+Radio+broadcasts%2C+mp3>

¹⁵ <http://wiki.opf-labs.org/display/SP/SB+Hadoop+Platform>

¹⁶ <http://www.myexperiment.org/workflows/4080.html>

¹⁷ <http://openplanets.github.io/scape-xcorr-sound/>

¹⁸ <http://www.scape-project.eu/deliverable/d16-1-lsdr-executable-workflows-for-experimental-execution>

¹⁹ <http://wiki.opf-labs.org/display/SP/SB+Experiment+SO4+Audio+mp3+to+wav+Migration+and+QA+Workflow>

- Validating that the migrated file is a correct file in the required format using JHOVE2
- Convert the MP3 file to WAV using mpg321
- Compare the two WAV files using xcorrSound waveform-compare

The new experiment described below is implemented to run on a Hadoop Cluster. It uses a Taverna workflow, which invokes a number of MapReduce jobs on the Hadoop cluster.

The Taverna workflow (see Figure 4.2.1-1, below) contains Hadoop jobs for the following tasks:

- Migration from MP3 to WAV using FFmpeg
- Convert the MP3 file to WAV using mpg321
- Compare the two WAV files using xcorrSound waveform-compare

The workflow does not have Hadoop jobs for:

- Validating that the migrated file is a correct file in the required format using JHOVE2. The value of this component is questionable, given the performance and reliability issues of JHOVE2.
- Extracting and comparing properties of the original and the migrated files using FFprobe. This component is very important and should be added to future iterations of the workflow. The performance is good enough that we are not worried that this component will impair the scalability of the workflow.

The project containing this workflow and associated Hadoop code is available from:

<https://github.com/statsbiblioteket/scAPE-audio-ga>.

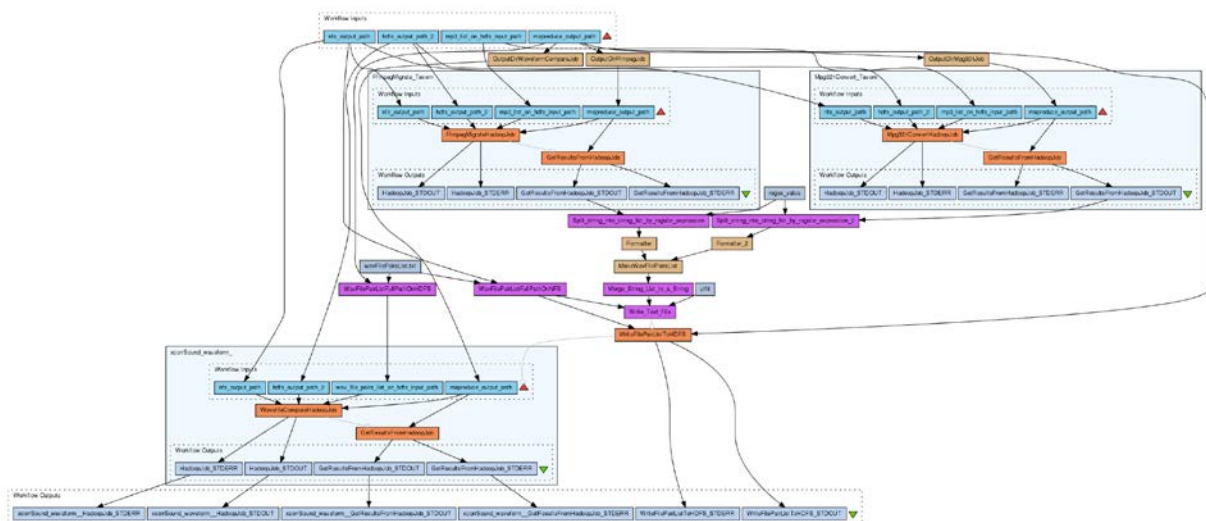


Figure 4.2.1-1: Taverna workflow for large scale migration of audio files

Taverna Workflow

In summary, this workflow carries out migration, conversion and content comparison.

The top left box (indicating a nested Taverna workflow) migrates a list of MP3 files to WAV files using a Hadoop²⁰ MapReduce job using the command line tool FFmpeg²¹, and outputs a list of migrated WAV files.

²⁰ <http://hadoop.apache.org/>

The top right box represents a conversion of the same list of MP3s to WAV files using another Hadoop MapReduce job which uses a different command line tool: `mpg321`²², and outputs a list of converted WAV files. The Taverna workflow then combines the two lists of WAV files and the bottom box receives a list of pairs of WAV files to compare.

In the bottom box the content of the paired files using a Hadoop MapReduce job using the `xcorrSound waveform-compare` command line tool, and the results of the comparisons are returned as outputs. The workflow takes lists of MP3 files as input, and currently the full list of files is first migrated and converted and the full list of pairs of migrated and converted files is then compared.

Input/Output

The file containing the list of MP3 files to be migrated is available on HDFS. The MP3 files are stored on NFS and the resulting WAV files are written to NFS. This has a number of reasons:

- The audio tools were written to read from and write to ordinary file systems.
- At SB digitally preserved material does not reside on HDFS, which means that in order to migrate from and to HDFS, we would first need to copy the MP3s to HDFS and later copy the WAVs from HDFS. These extra copy operations are expensive, when we are talking large-scale audio collections.
- The SB Hadoop Platform is set up using network storage as local storage, which means that we do not exploit the HDFS locality property, and thus accessing the files on NFS rather than HDFS does not present a large overhead.

All preservation events, e.g. properties of original and migrated files were compared and accepted, and all log files are all written to HDFS. This means we have a rather complex input/output model with input from both HDFS and ordinary file systems, and also with output to both HDFS and ordinary file systems. If this workflow were to be used in production, repository integration²³ would need to be added, such that data can be both retrieved from the repository and written to the repository.

Future Work

What we would like to do next is:

- Run an experiment using 1TB of MP3 files on the SB Hadoop cluster. This however requires some updates to the workflow. For 1TB input MP3 files, the workflow currently generates approximately 25TB of output and temporary WAV files.
- Extend the workflow with property comparison. The `waveform-compare` tool only compares sound waves; it does not look at the header information. This should, however, be part of a quality assurance of a migration. The reason this is not top priority is that FFprobe property extraction and comparison is very fast, and will probably not affect overall workflow performance much. The reason this has not been done yet is again sparse resources.

²¹ <http://www.ffmpeg.org/>

²² <http://mpg321.sourceforge.net/>

²³ <http://www.scape-project.eu/deliverable/d8-1-recommendations-for-preservation-aware-digital-object-model>

4.3. User Story: Large Scale Image Migration

This user story is:

As a curator of image files, I need a digital preservation system that can migrate a large number of images from one format to another, ensuring that the migrated images conform to our institutional profile, that no image data is lost and that the migration is cost effective (saving storage for example).

It describes a requirement whereby a curator needs to perform a format migration on a large set of data. The following user requirements were identified:

1. We need to be able to migrate TIFFs to JPEG2000s
 - i. Ideally we can migrate TIFF to a JPEG2000 conforming to *any profile within the limits of the JPEG2000 standard*
 - ii. Migration must support the recommended JPEG2000 profile
2. We need to compare a JPEG2000 image file technical metadata and profile to the recommended profile.
3. We need to ensure that the JPEG2000 contains all of the image data
4. We need to ensure that the JPEG2000 is a good and complete copy of the TIFF
5. We need to be able to report on the storage saving and perhaps cost benefit of doing this

Details about specific experiments relating to this user story are described below.

It is worth noting that it would be possible to change the target file format for the migration quite easily - with one step (Jpylyzer) that would need to be changed for different format validation, and possibly some small changes in the reporting step. In one experiment below the codec is substituted for another JPEG2000 one very easily, as the codec is for the same file format.

4.3.1. Experiment: KB Metamorfoze Image Migration & QA

Dataset	National Library of The Netherlands (KB) Metamorfoze (sample batch)
Platform	KB SCAPE Platform (pseudo-distributed Hadoop + SCAPE tools)
Workflow(s)	A Java workflow and a batch file workflow

The Metamorfoze digitization programme started digitizing for conservation purposes in 2007. In spite of the KB's policy to store only JPEG2000 master files, Metamorfoze has stored all images in TIFF format. Accordingly, it was decided to migrate the whole Metamorfoze collection to JP2 format (JPEG2000 Part 1). The KB received a total of approximately 146 TB in TIFFs, which will be converted to JP2s with an expected total size of 73 TB, thus significantly reducing storage costs for the long-term preservation of the scanned images. The migration is currently performed on the level of separate batches that are processed one at a time and in sequential order using a batch script.

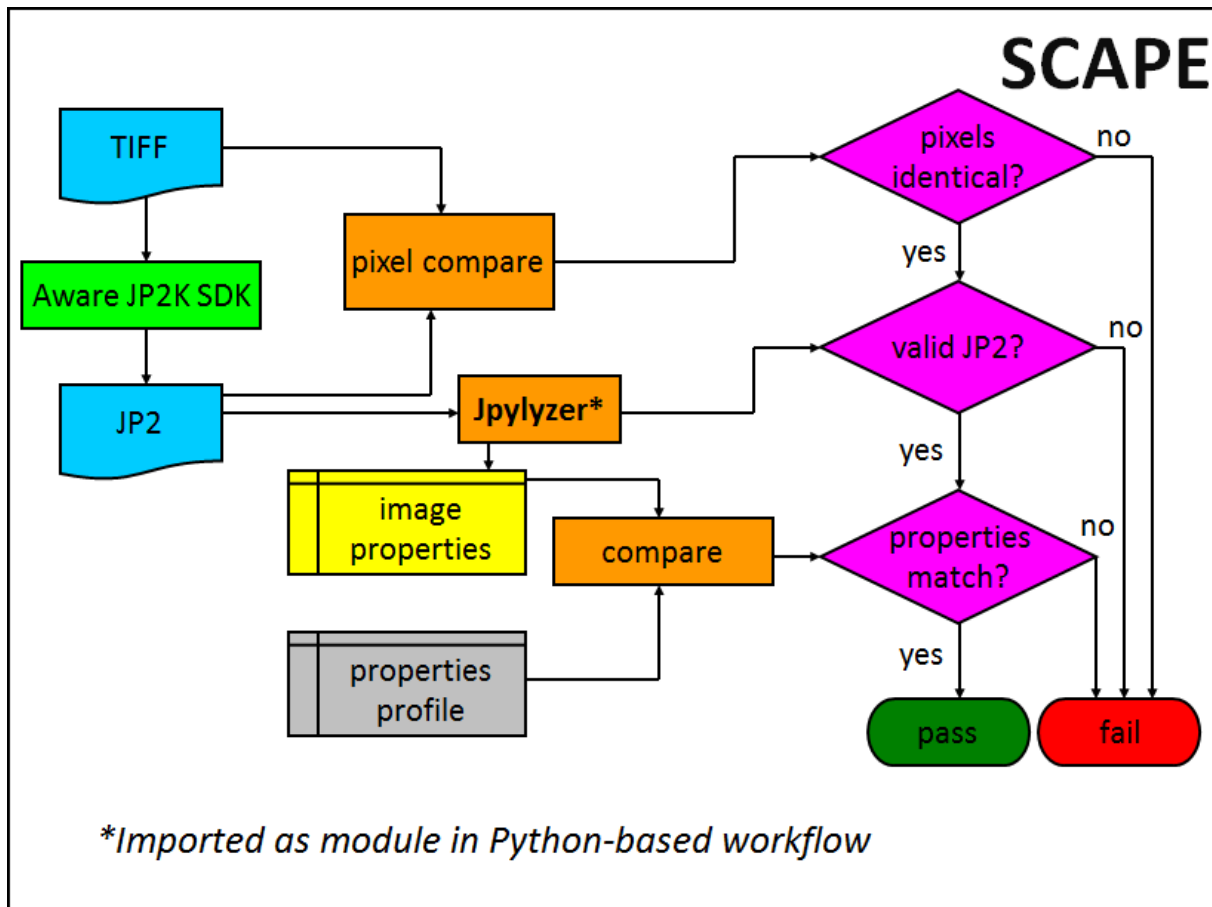


Figure 4.3.1-1: Diagram of the workflow in the Metamorfoze migration project

Conceptual workflow

For the operational migration, a custom Python script (*MMBatchConverter*) is used that, for each batch, converts all images, updates the associated metadata, and finally runs a number of quality checks. Specifically, it performs the following actions for each TIFF image:

1. Extract the capture metadata to a sidecar file in XMP format (using ExifTool)
2. Convert the TIFF image to lossless JP2 according to a pre-defined profile. The metadata sidecar files described in the previous step are embedded in the JP2. The conversion uses the Aware JPEG2000 SDK²⁴, which is called through a Python wrapper.
3. Convert the newly created JP2 back to a (temporary) TIFF using Kakadu's *kdu_expand* tool²⁵. This step is needed for the pixel comparison (see below), since ImageMagick's²⁶ support of JP2 is problematic, due to its use of an older JPEG2000 library.
4. Do a comparison of the pixel values in the source TIFF and the temporary TIFF using ImageMagick. Count the number of non-identical pixels (which must be 0, since we're using lossless compression).

²⁴ <http://www.aware.com/imaging/jpeg2000sdk.html>

²⁵ <http://www.kakadusoftware.com/>

²⁶ <http://imagemagick.org/>

5. Run Jpylyzer on the JP2 to check its validity, and to get the image's technical characteristics.
6. Compare the image characteristics against the pre-defined profile.
7. An image passes the QA if all pixels are identical, the output is valid JP2 and the technical characteristics are consistent with the profiles.

In addition to the above steps (which are repeated for each individual image), the *MMBatchConverter* script updates the concordance tables that contain the structural metadata of each batch. After all images in a batch are converted, the script does two cross-checks that verify if all images that are defined in the concordance table actually exist, and if all images in the batch are defined in the concordance table. The script also computes MD5 checksums for each created JP2, which facilitates the detection of any changes at a later stage. The results of the conversion and the outcome of all quality checks are then written to a number of log files.

Converting the entire Metamorfoze collection (146 TB in TIFFs) to JP2 using the *MMBatchConverter* script is expected to take approximately 8000 hours of machine time if run on a KB DMZ workstation²⁷.

Implementation for SCAPE

For the purpose of SCAPE, an implementation of the above workflow was done in Java as to fully leverage Hadoop functionalities and components provided by the SCAPE platform. The Java implementation tries to emulate as closely as possible the operational workflow in the *MMBatchConverter* script. The source is available on Github²⁸.

In this implementation, the migration is triggered via the execution of a shell script that invokes the main class of the Java project. As all the business logic could be implemented in the main Java class, it was decided to not also provide a Taverna workflow. Such a workflow would have simply consisted of a wrapped call to the conversion script as a local tool command.

The test was executed on the KB SCAPE platform consisting of a (pseudo-distributed) Hadoop cluster with 4 nodes (1 master, 3 workers), each running 1 hyper threading CPU core on a virtualized Ubuntu Linux 12.04 server. The Cloudera CDH4 was chosen as the Hadoop distribution. The following software had to be pre-installed on each node:

1. jpwrapppa: a Python wrapper for the command-line tool of the Aware JPEG2000 SDK
2. kdu_expand: a command-line tool from the Kakadu JPEG2000 Toolkit
3. Jpylyzer: a validator and feature extractor for JP2 images
4. Exiftool: a command-line application for reading and writing image meta information
5. GraphicsMagick: a software suite to create, edit, analyse or convert bitmap images
6. Probatron4J: a Java tool for checking XML content using ISO Schematron schemas

Data storage methods

The sample dataset for SCAPE consists of one batch from the ongoing Metamorfoze migration, containing 8047 single-page colour TIFF images and adding up to approx. 170 GB of data. In addition to the images, descriptive as well as technical metadata (DMD) and the log files of the migration of the particular batch in the operational Metamorfoze migration project have also been collected for

²⁷ The machine is a HP proliant ML370 G6 with 2x Quad core 3 Ghz and 32 Gb memory that is connected to the SAN through an 10-40 GB connection. Three disks are reserved on the SAN for performing the conversion.

²⁸ <https://github.com/KBNLresearch/hadoop-jp2-experiment>

comparison. While the regular Metamorfoze migration uses SAN storage, the sample batch has been ingested into HDFS for processing with the KB Hadoop platform²⁹.

Findings

The KB has started the Metamorfoze migration project in April 2012. A total of approx. 4.7 million TIFF images or 147TB of data is expected to be processed using this workflow. As of April 2014, the migration is still ongoing with about 60% of the total amount already converted.

The purpose of the SCAPE experiment was to determine in how far this workflow could benefit from parallelization in order to increase throughput but also standardisation and to ease quality assurance with the help of SCAPE architecture and tools. However, since the workflow as defined requires the invocation of several local command line tools, this could not be implemented in an optimal way so that full advantage could be taken from using Hadoop as the execution engine. Local command-line tools are spawned as individual Mappers, which causes some overhead compared to, for example, an implementation that would be done fully in Java and thus native to Hadoop.

However, in comparison to the expected 8000 hours on the KB DMZ workstation with the *MMBatchConverter* script, on a pseudo-distributed Hadoop cluster running on a single virtual server with similar specs as the KB DMZ workstation, the time required to do the conversion could in principle be reduced to roughly 5400 hours when the Hadoop implementation is used.

4.3.2. Experiment: BL Newspapers on the BL Platform

Dataset	BL 19th Century Digitized Newspapers
Platform	SCAPE Platform
Workflow(s)	Three equivalent workflows; A Taverna Workflow, a Java workflow and a batch file workflow

This experiment consolidates two scenarios from D16.1 - LSDRT2 and LSDRT3. They evolved to the point that they were exactly the same, but one had the addition of an arguably essential step; checking the migrated image data versus the original. Workflows were developed to test different methods of execution (with combinations of Taverna and Hadoop, just Hadoop, and plain batch files). After the initial testing, the experiments were extended to add retrieval of files from different repositories/file stores.

The dataset contains master TIFF files (greyscale), access files, and metadata. For this experiment we make use of 41,963 master TIFF files; approximately 1TB of data.

The test platform used was a 29-node virtualised Hadoop cluster running Ubuntu Linux and the Cloudera distribution of Hadoop (Version: CDH4). There are 28 nodes for job execution, each with 1 CPU. The following software was pre-installed on the nodes:

1. Taverna command line: command line execution of Taverna workflows
2. Jpylyzer: JP2 (JPEG2000 Part 1) validator and properties extractor

²⁹ <http://wiki.opf-labs.org/display/SP/KB+Hadoop+Platform>

3. Matchbox: image comparison QA tool (amongst other features)
4. Kakadu: a JPEG2000 codec
5. OpenJPEG: a JPEG2000 codec
6. ImageMagick: image comparison QA tool (amongst other features)
7. Exiftool: an image metadata extraction tool

Where no operating system packages were available, or a more recent version was required, software was distributed to a local user's home directory on all of the nodes via a simple shell command. Distributing software in this manner meant that no files at the operating system level were affected, thus maintaining OS stability.

An attempt was made to use the SCAPE ToolWrapper³⁰, and a tool specification xml file containing the definition for the correct encoding parameters was created. However, this effectively pushed a single command beneath another layer of abstraction so instead of adding the overhead, the command was used directly instead. The jp2check³¹ library used in the Java workflow generates command line encoding settings for different codecs, and also use Schematron to check the outputs from Jpylyzer for verification of the JPEG2000 encoding profile.

Data storage methods

Copies of the 1TB dataset were stored and accessible from the following repositories:

1. The HDFS local to the Hadoop cluster
2. A Webdav³² enabled NAS device local to the Hadoop cluster
3. A Fedora-Commons v3 repository local to the Hadoop cluster (in a VM, with data storage on the NAS device)

Conceptual workflow

This is a description of the workflow that is implemented in different ways; those methods are detailed later on.

1. Metadata extraction from original image

The first step is to extract the metadata from the original TIFF file. For this we use Exiftool to extract both Exif and TIFF metadata as an XML file.

2. Image migration

We use a JPEG2000 codec to migrate an image from TIFF to JP2 (JPEG2000). The primary codec we have been using during is OpenJPEG, but some tests were run using Kakadu. The encoding profile we used was the British Library newspaper profile³³. There was an issue with OpenJPEG when using one of the settings in the profile, this bug was reported and should be fixed in a future release³⁴.

³⁰ <http://openplanets.github.io/scape-toolwrapper/>

³¹ <https://github.com/bl-dpt/jp2check>

³² <http://en.wikipedia.org/wiki/WebDAV>

³³ <http://wiki.opf-labs.org/display/JP2/Example+JP2+profiles>

³⁴ <http://code.google.com/p/openjpeg/issues/detail?id=209>

3. Metadata extraction from migrated image

Following a successful migration image metadata is extracted from the JP2 file using Exiftool, as described above, and also with Jpylyzer.

4. Image comparison

This step of the workflow compares the image payload (what you see) of the original and migrated files. This step is important as it validates that the actual image has migrated successfully and can be decoded from its new format.

5. Reporting

After all the previous stages of the workflow are completed, some reporting on the work completed is made:

1. Status of image payload comparison* This also implicitly checks the dimensions of the image)
2. Whether the JP2 file is valid*
3. Whether the encoding profile matches the one that was requested*
4. Zip all output files, along with a report XML file, into a BagIt³⁵ like package

Note that steps marked * are not executed in the batch workflow.

Comparison tools for image migrations - Matchbox vs ImageMagick

After the initial testing it was apparent that using Matchbox for image format migration checks was outside its use case as it was not well suited to detecting minor/subtle differences between an original and compressed image. Indeed, Matchbox is designed to find duplicates in the content within sets of images, where two images with the same content can have major differences between them (e.g. one is rotated, in a different resolution, or one has an additional border). For an image migration it is important to be able to find out whether the image data is the same (if using lossless compression) or as close to the original as possible (if using lossy compression).

Instead of Matchbox we made use of ImageMagick's `compare` command; it can generate various different metrics for comparison of images. We found that peak signal-to-noise ratio (PSNR)³⁶ was a useful metric to use; however, there are many other metrics that could have been used³⁷.

Differences between OpenJPEG and Kakadu JPEG2000 codecs

When analysing the results from the initial large scale testing we found that there were differences between the OpenJPEG and Kakadu JPEG2000 codecs. When using lossy compression the two main differences were speed and image quality. A run against the 1TB dataset was significantly faster when Kakadu was the codec (17h25m), in place of OpenJPEG (57h02m). However, the image quality was slightly lower for Kakadu (as measured by PSNR) - all successful migrations with OpenJPEG were over 50dB PSNR, and this threshold was lowered to 48dB for a successful Kakadu run. It could be that a slightly higher threshold would have sufficed but this was not tested. As a result of these findings

³⁵ <http://en.wikipedia.org/wiki/BagIt>

³⁶ http://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio

³⁷ <http://www.imagemagick.org/Usage/compare/>

we looked at this issue further and produced a paper that investigates this matter. This paper was presented at iPres 2013³⁸.

Workflow execution methods evaluated

Initial workflows using the following execution methods were tested:

1. Batch file execution (note that this method does not include any reporting steps)
2. Java defined workflow (i.e. a native MapReduce job)
3. A MapReduce job that executed one Taverna workflow per input file
4. *A Taverna workflow controlling several MapReduce jobs (one per processing step)**
5. *A MapReduce job that submitted workflows and files to a Taverna Server+*

*Testing was run for an additional workflow type, where a Taverna workflow controlled a series of distinct Hadoop MapReduce jobs. The set up for this job was complex and the implementation required state to be kept in a message queue between the MapReduce jobs which added further complexity. If one of the steps failed, for just one of the files, then this implementation would terminate early, in a difficult-to-restart state. This work was not taken further after this experiment due to the workflow being very complex and the apparent restart issues with this implementation. These issues are not necessarily insurmountable but they do add a degree of complexity that does not exist in other workflow execution methods tested, for little apparent reward.

+Following testing we found that Taverna Server was not suitable for use in the way in which it was used in this workflow. It would have required a separate installation on each node in the Hadoop cluster. However, the main issue with this method of execution was that it did not keep a copy of Taverna in memory for reuse by workflows and instead started a whole new instance of Taverna for each submitted workflow. As this was in addition to the main Taverna Server process it added more overhead to the execution rather than less. After contacting the developers of Taverna Server it was clear that this was the intention (to avoid unwanted interactions between workflows) and so this method of execution was not developed further.

Having discounted the two methods as described above we continued to refine the remaining workflows and testing them with the full 1TB dataset.

Detailed description: Batch workflow

The batch workflow method is written as a Bash shell script for Linux and was designed to replicate as much of the conceptual workflow as possible, to provide a baseline for execution on a single node of the Hadoop cluster, without the additional overhead of Hadoop. The batch workflow does not perform any additional reporting steps on the tool outputs, as indicated above. This could be added, however, the anticipated execution time for checking an XML file etc. is not significant and was not deemed to be worthwhile.

“Chutney” Hadoop wrapper³⁹

To enable a like-for-like comparison as much as possible, a MapReduce program called Chutney was created that is responsible for recovering files from storage, placing them in a local directory,

³⁸

http://purl.pt/24107/1/iPres2013_PDF/An%20Analysis%20of%20Contemporary%20JPEG2000%20Codecs%20for%20Image%20Format%20Migration.pdf

³⁹ <https://github.com/bl-dpt/chutney-hadoopwrapper/>

executing one of the workflow types, storing the workflow outputs and generating the output for the Reduce phase.

A by-product of this design is that the workflows can be executed without using MapReduce and therefore do not need to contain any understanding of MapReduce. Thus the workflows are simple and self-contained and can be easily interchanged.

It may be possible to use the self-contained workflows for assessing the batch workflow execution time, and we will bear this in mind for future testing.

Detailed description: Java workflow

The Java workflow directly controls the execution of the tools as described in the conceptual workflow. Where possible it executes code in Java - in this instance it is just the JP2 profile check and report generation code. The non-Java, external tools that are used within this workflow are executed by the Java code.

Tools are executed with the workflow directly calling the command line applications from Java, which are executed in a separate process. Further integration would be possible by using the OpenJPEG Java JNI bindings, for example, but this was not implemented due to the state of the interface. Additionally it may be possible to use Jpylyzer (written in Python) directly in the Java code by using a tool such as Jython. However, this approach was not tested. The anticipated additional speed boost for a short runtime piece of code versus the time taken for an image encode is not significant. It also ties the tools more closely to Chutney making it more difficult to introduce new versions of the tool.

Detailed description: Taverna workflow

The Taverna workflow has evolved over time to its current state. The current workflow takes three inputs;

1. A compiled Schematron file for validating the JP2 profile used for encoding
2. A TIFF file
3. The original name of the TIFF file

To note: Taverna workflows are graphs made of several "Services" (nodes) that are linked together using "Data links" (vertices).

The workflow has to be passed a TIFF file and the name of the file separately due to how Taverna passes data between steps. The file data is transferred between workflow services transparently by Taverna and named as specified in the service configuration ("File inputs" & "File outputs"). It is necessary to pass a binary blob as otherwise a file reference would have to be passed, complicating the execution. The original filename that is passed to the workflow is used in a BeanShell service to correctly name the files in the output zip file.

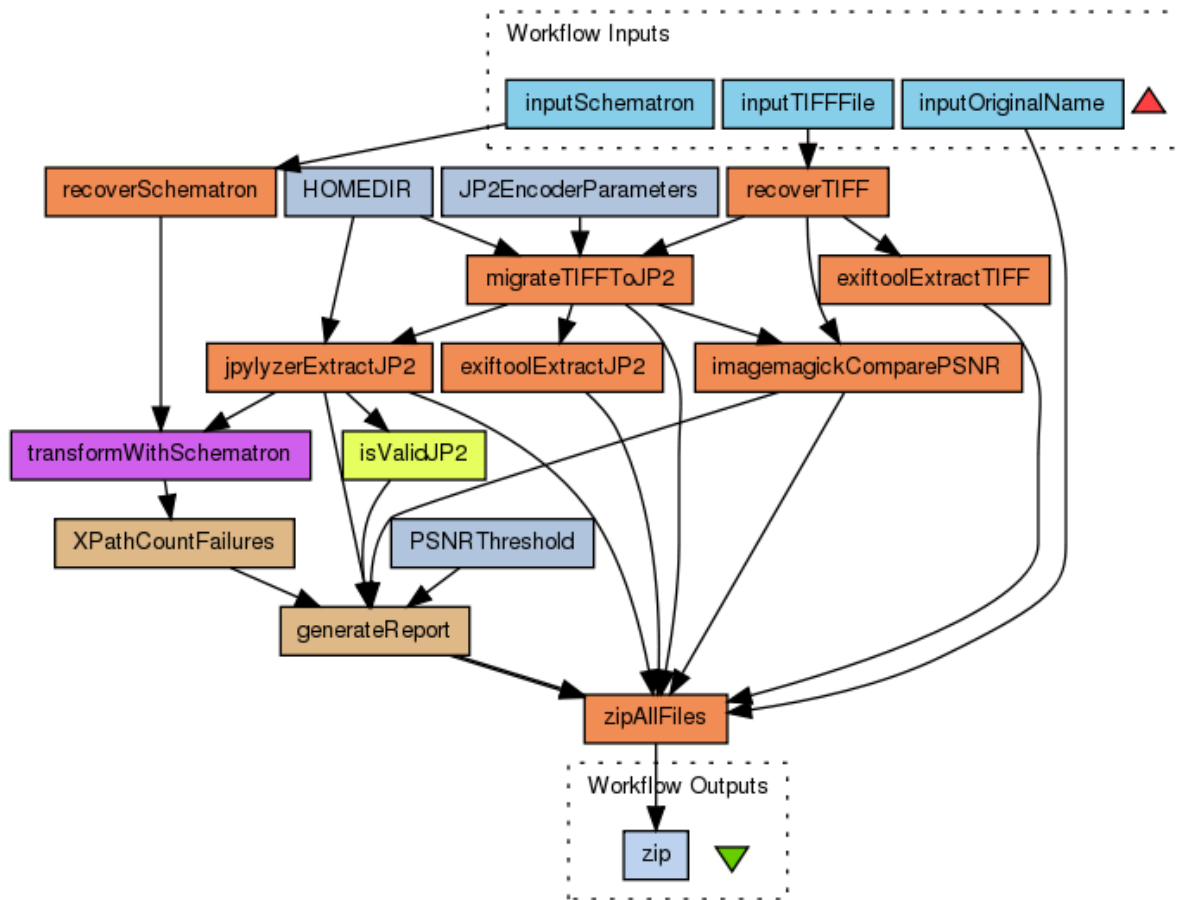


Figure 4.3.2-1: Taverna Workflow for TIFF to JP2 migration⁴⁰

This workflow has several interconnected services. Taverna ensures that the order of execution of the services is correct and parallelizes it by running independent services together wherever possible.

The Workflow Input boxes at the top (light blue) represent inputs to the workflow and the Workflow Outputs box (blue) at the bottom represents the workflow output. The orange boxes such as recoverSchematron and recoverTIFF are tool services, and they execute an external program. The remaining boxes are dealt with inside Taverna; blue: constant values such as isValidJP2 are yellow: built in Taverna BeanShell services like XPath execution, such as transformWithSchematron, are pink and custom BeanShell services such as generateReport are brown.

The orange tool services largely just execute a single command. The services define the filename that the input and output files will use, Taverna is responsible for moving the data around. The tool can assume that the input file will exist i.e. "input.tif", and is responsible for ensuring that the output file is created, i.e. "input.tif.jp2". Taverna can then reuse that output file as an input to any linked services. The "zipAllFiles" service ensures that the output files are renamed according to the correct input filename and generates a BagIt-like output zip.

⁴⁰ <http://www.myexperiment.org/workflows/3401.html>

This version of the workflow uses much more of the built-in Taverna functionality than earlier development iterations and is simpler and more understandable as a result.

Tool installation and use

All of the workflows described above require the installation of additional tools on the cluster. There were three ways to deal with this;

1. Install operating system packages (official or SCAPE project ones) - wherever possible this was the case but not all tools were available, or the latest versions of the tools.
2. Locally compiled binaries, or other packages of binaries, copied to a home directory on all the nodes in the cluster. This can be achieved with a couple of lines of shell script. In this case the binaries do not interfere with the operating system files and are available for use in the workflows.
3. Binaries can be embedded into the Chutney jar file and extracted when required. This is useful as it can be done without requiring administrator access to the cluster and ensures that the workflows are self-contained. However, it means the jar file can get much larger and there is a setup cost to preparing the tool for execution. The number of times this has to happen might make this very costly. If this happened once per Mapper, and that Mapper processed thousands of records, then this may be mitigated somewhat.

For this experiment we use the first and second methods above, with some data being embedded into the jar similar to the third method, such as the Taverna workflow and Schematron files. This decision was due to the expected reuse of the tools during development of the workflows, so that they did not have to be re-copied throughout the cluster whenever a run was started.

Data storage methods compared

An interesting result was encountered during testing: the total runtime of the workflow was not significantly affected by the location of the data; having the data locally in HDFS was not significantly faster than recovering the data from a repository and saving it back out to the repository. It is worth noting the following points:

1. The method for passing the files to the workflow did not enable Hadoop to make use of HDFS data locality, although all the data was held internally within the cluster.
2. Accessing and saving files from an external (but also local) Fedora Commons repository took 57h50m vs 57h02m for accessing from HDFS. This was not a significant amount of time. Copying files from an external (but also local) NAS into HDFS took 08h03m - factoring this into the execution time means that it took much longer than just directly accessing the files remotely. That time does not include copying the migrated files from the cluster back to the NAS. The same NAS also hosted the data for the Fedora Commons repository.
3. These findings are probably due to the long execution time of the image migration workflow and would not be applicable for cases where the ratio of execution time of the workflow is less compared to the input data, i.e. a traditional MapReduce type text-processing workflow.

4.4. User Story: Large Scale Ingest

The User Story is:

As an institution we need a system that will enable us to ingest a large number of digital objects and associated metadata into our digital repository securely, correctly and with acceptable performance so that we can ensure safe deposit of this data.

Due to the increasing amount of digital objects libraries, research institutions and digital archives must handle, the scalability and performance of repositories is becoming more and more important.

Scalability in large scale digital object repositories depends on a variety of parameters such as the number, size, complexity, and heterogeneity of objects. Therefore, in relation to digital preservation, any data processing, such as quality assurance, technology watch, preservation planning, and the digital object repository must itself be scalable.

4.4.1. Experiment: Large Scale Ingest with Fedora4

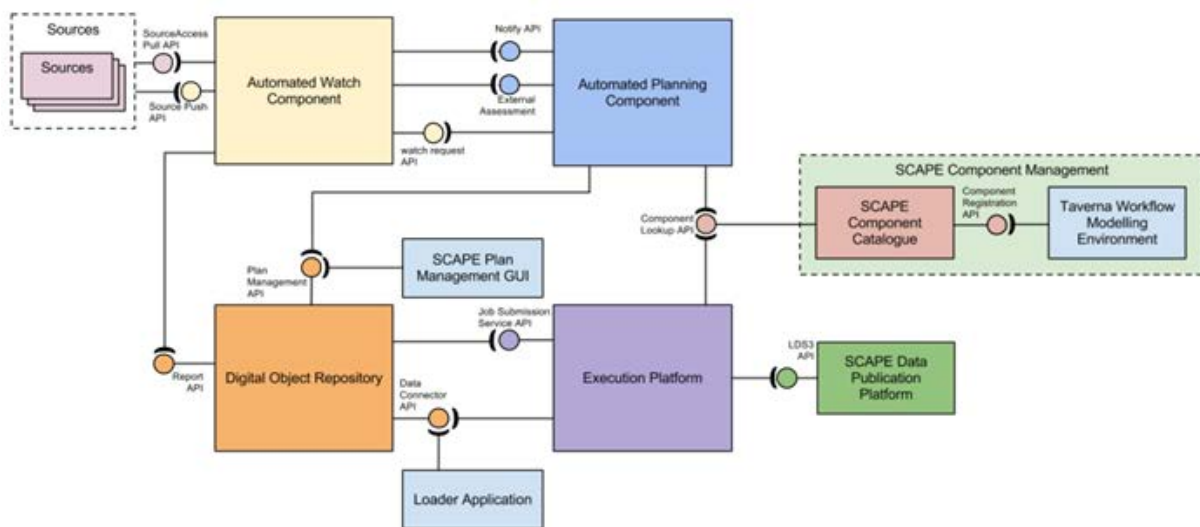


Figure 4.4.1-1: Overview of the SCAPE architecture.

As illustrated in figure 4.4.1-1, the SCAPE Platform consists of several services such as the Execution Platform, Planning and Watch Components and the Digital Object Repository. Several repository implementations are being used in the project, such as the Fedora 2 based RODA repository by Keep Solutions, the Fedora 3 based DOMS repository by SB, the Rosetta repository by ExLibris and a Fedora 4 based implementation created by FIZ. To integrate repositories into the SCAPE environment the following APIs have been defined:

1. Connector API: CRUD operations for digital objects.
2. Plan Management API: Interface to manage and execute preservation plans.
3. Report API: Interface to retrieve information about events taking place inside a repository, e.g. ingest.

Ingesting a large amount of data into a repository is achieved by using a Loader Application that is reading data from the file system and is capable of monitoring the ingest progress and creating reports about it. The Loader Application is designed to be used by any repository that exposes the Data Connector API and removes the burden from a repository provider to implement a specific Loader Application for their repository.

The Austrian National Library's use case of ingesting a digital book collection was used to design the scalable ingest process. The library receives hundreds of scanned books every month, and therefore being able to ingest those digital books in the given time frame into a digital object repository is required. The individual book pages of this collection are available as JPEG2000 files with corresponding full-text and HTML full-text including layout files. The digital books are represented by METS⁴¹ which aggregates the pages of the digital book. Each page of the book is represented by an image, html file and plain text file with references to the physical files on a file server.

The digital object repository is used to store the metadata of the book scans and it does not manage the binary files of each book. Other Repositories like RODA and ROSETTA are only able to deal with managed content, as opposed to the referenced content used here. Additionally, the managed storage option has been looked into during the ingest tests.

Fedora 4, the repository used in this use case, is still under development by Duraspace and SCAPE partner FIZ Karlsruhe. The tests performed are based on alpha releases of Fedora 4 which is built on top of:

- Modeshape⁴², a JCR repository maintained and developed within the JBOSS community
- Infinispan⁴³, a distributed cache implementation
- JGroups, a messaging toolkit to transfer states between nodes.

All of these components can be configured while setting up a cluster. The configuration of each of these layers adds complexity into the overall scenario and is outlined briefly in the following.

Fedora 4 Cluster Topologies

Basically clustering can be configured in two distinct topologies handled by Infinispan: replication and distribution. Replication mode means that all entries are replicated to all nodes and offers high durability and availability of data. This clustered mode provides a quick and easy way to share state across a cluster, however replication practically only performs well in small clusters, due to the number of replication messages that need to happen as the cluster size increases.

Replication can be synchronous or asynchronous. Synchronous replication blocks the caller until the modifications have been replicated successfully to all nodes in a cluster. Asynchronous replication performs replication in the background. Infinispan offers a replication queue, where modifications are replicated periodically and can therefore offer much higher performance as the actual replication is performed by a background thread. Asynchronous replication is faster, because synchronous replication requires acknowledgments from all nodes in a cluster that they received and applied the modification successfully (round-trip time).

⁴¹ <http://www.loc.gov/standards/mets/>

⁴² <http://www.jboss.org/modeshape>

⁴³ <http://infinispan.org/>

The distribution mode replicates the entries only to a subset of the nodes in the cluster. Compared to replication, distribution offers increased storage capacity, but with reduced availability (increased latency to access data) and durability. Distribution makes use of a *consistent hash*⁴⁴ algorithm to determine where in a cluster entries should be stored and is configured with the number of copies each cache entry should be maintained cluster-wide. Number of copies represents the trade-off between performance and durability of data. The more copies are being maintained, the lower performance will be, but also the lower the risk of losing data due to server outages.

Beside those cluster topologies, Infinispan can also be configured in a local mode to run as a single instance. Running Fedora 4 in a local mode restricts the scalability approach only to vertical scalability and horizontal scalability is not given. In this use case the focus was on horizontal scalability and the local mode was used only as a reference to compare the performance of clustered modes with the local mode of Fedora 4.

Hardware Clusters

To test the horizontal scalability of Fedora 4 we have been able to use several hardware clusters at FIZ Karlsruhe, at the Steinbuch Center for Computing at KIT (SCC), at Amazon AWS, and at University of Timisoara, Romania – one of the new partners in SCAPE.

To distribute the software on the distinct cluster nodes shell and puppet scripts to deploy the necessary software and to start and stop the processes on the nodes were developed. Also, a benchmark tool was developed and performance tests were carried out using JMETER. A summary can be found on the Fedora 4 Wiki⁴⁵ and the Open Planets Wiki⁴⁶

Cluster Performance Issues

When it comes to write operation, e.g. performing a large scale ingest, there are four things that are considered to be critical and which will be examined. These are, in order of cost:

- Network communication.
- Marshalling.
- Writing to the cache store.
- Locking, concurrency and transactions.

In the following each of these four items will be briefly discussed.

Network Communication

Data can be propagated to other nodes in a synchronous or asynchronous way. When synchronous, the sender waits for replies from the receivers and when asynchronous, the sender sends the data and does not wait for replies from other nodes in the cluster. With asynchronous modes, speed is more important than consistency. For the network part, JGroups can be configured to send any request across the network but will not wait for a reply from the receiver.

⁴⁴ http://en.wikipedia.org/wiki/Consistent_hashing

⁴⁵ <https://wiki.duraspace.org/display/FF/Fedora+4.0+Alpha+3+Release+Notes#Fedora4.0Alpha3ReleaseNotes-Benchmarking>

⁴⁶ <http://wiki.opf-labs.org/display/SP/Ingest+of+digitized+book+METs+into+Fedora+4>

Marshalling

Asynchronous marshalling means whether the actual call from Infinispan to the JGroups layer is done on a separate thread or not, i.e. requests can return back to the client quicker compared to synchronous marshalling. The downside is that client requests can reach the JGroups layer in a different order in which they're called. This can effectively lead to data inconsistency issues in applications making multiple modifications on the same key/value pair.

Cache Stores

Infinispan ships with several cache loaders that utilize the file system as a data store.

- FileCacheStore - a simple file system-based implementation
- BdbjeCacheStore - a cache loader implementation based on the Oracle/Sleepycat's BerkeleyDB Java Edition.
- JdbmCacheStore - a cache loader implementation based on the JDBM engine, a fast and free alternative to BerkeleyDB.
- LevelDBCachedStore - a cache store implementation based on Google's LevelDB, a fast key-value store.

For all the tests in this scenario we have been using the FileCacheStore and LevelDBCachedStore option.

Locking, Concurrency and Transactions

With optimistic transactions locks are being acquired at transaction prepare time and are only being held up to the point of the transaction commit or rollback. Optimistic transactions should be used when there is *not* a lot of contention between multiple transactions running at the same time.

Pessimistic transactions obtain locks on keys at the time the key is written and might be a better fit when there is high contention on the keys and transaction rollbacks are less desirable. Pessimistic transactions are more costly: each write operation potentially involves a RPC for lock acquisition.

Deadlocks on the other hand can significantly reduce the throughput of a system, especially when multiple transactions are operating against the same key set.

Transactions cannot be completely disabled since Modeshape relies on a working transaction configuration on the Infinispan layer. Therefore only the option to use pessimistic vs. optimistic locking can be tested.

Summary

We have not only tested Fedora 4 but also the underlying Modeshape repository to get a better understanding of the behaviour of the different layers Fedora 4 is made of. To evaluate the clustering functionality of the above stack described above, the following setup was tested:

- Modeshape (without Fedora 4 on top) with Infinispan and JGroups
- Fedora 4 with Modeshape, Infinispan and JGroups
- Fedora 4 with the SCAPE Data Connector API implemented

Parameters that are crucial for the cluster performance such as the cluster topology, the network communication, the CacheStores were varied and different transaction models have been tried. Shell scripts and puppet scripts to deploy a Fedora cluster efficiently on a hardware cluster were developed along with benchmarking tools and JMETER tests. Different hardware clusters with different hardware in terms of CPU, disk I/O, and network bandwidth were used. As a result it was not possible to find a cluster configuration that satisfies the SCAPE use case requirements. In fact, a huge drop in the performance of a Fedora 4 cluster (and Modeshape cluster), by a factor of 100 compared to a single, non-clustered installation was observed, independent of the configuration and hardware used. . The results have been discussed with Duraspace and they decided to postpone the release of the Fedora 4 clustering feature to Fedora 4.1 which is beyond the SCAPE projects end date.

4.5. User Story: Policy-Driven Identification of Preservation Risks in Electronic Documents

The user story is:

Digital repositories typically hold large numbers of electronic documents from various sources. Common document formats such as PDF and EPUB include features that are potential risks for long-term accessibility and preservation. Hence, in order to sustainably manage their collections, institutions may want to identify specific preservation risks, either at ingest or at some later stage.

4.5.1. Experiment: Validate PDF&EPUBs and check for DRM

Dataset	PDF files from the Govdocs1 corpus. There are 231,683 PDFs (127.8GB) in the dataset.
Platform	SCAPE Platform
Workflow(s)	A Java workflow contained within the DRMLint tool

This workflow is designed to detect the presence of DRM/encryption within PDF and EPUB files, and to test that the file formats are valid. Optionally, it can extract the text from the input files. It processes PDF files from HDFS and outputs a report XML file containing the results, inside a BagIt⁴⁷ style archive.

The workflow makes use of a software tool developed in the Preservation Components subproject; DRMLint. To detect validity and DRM, DRMLint makes use of several external software libraries along with its own internal methods.

The workflow for this experiment is a straightforward MapReduce Java program contained within the DRMLint tool itself. The input files are given to the tool as a list of files in HDFS. The analysis output from DRMLint is stored in HDFS and in addition, the output from the Reduce phase of the workflow

⁴⁷ <http://en.wikipedia.org/wiki/BagIt>

provides a list of comma separated values (filename, if valid, if DRM detected). The output from the Reduce phase can be analysed, along with the in-depth results in the output stored in HDFS.

Work is underway to integrate the pdfPolicyValidate⁴⁸ Schematron code into DRMLint for more fine-grained policy checks.

4.6. User Story: Quality Assurance of Digitized Books

The user story is:

As a cultural heritage institution, we need a digital preservation system that can identify whether there have been any cropping errors during the digitisation process.

4.6.1. Experiment: Quality Assurance of Digitized Books Experiment

Dataset	Austrian National Library - Digital Book Collection
Platform	ONB Hadoop Platform
Workflow(s)	http://www.myexperiment.org/workflows/3069

The goal of this experiment was to parse large amounts of HTML files that are part of a large book collection where each HTML page represents layout and text of a corresponding book page image. These HTML files have block level elements described by the HTML element <div>. Each element has a position, width and height representing the surrounding rectangle of a text or image block. The average block width of these <div> elements is used to detect quality issues that exist due to cropping errors.

In the SCAPE project, the Taverna Workflow Workbench⁴⁹ is used for orchestrating long term preservation tools and services operating on an underlying data flow. The first point of investigation was to find a way of chaining Hadoop jobs using Taverna’s Tool service invocation mechanism.

The cropping error detection workflow is an applicable scenario for *MapReduce* because it uses the *Map* function for parallelisation of the HTML parsing and the *Reduce* function for calculating the average block width. However, some data preparation is needed before the *MapReduce* programming model can be applied effectively.

The following diagram in figure 4.6.1-1 shows a Taverna workflow that combines several Hadoop job components to model a linear data flow. The Java code of the Hadoop job implementation for SequenceFile creation and hOCR parser are available on Github.⁵⁰

⁴⁸ <https://github.com/openplanets/pdfPolicyValidate>

⁴⁹ <http://www.taverna.org.uk/>

⁵⁰ <https://github.com/shsdev/sequencefile-utility> and <https://github.com/shsdev/hocr-parser-hadoopjob>

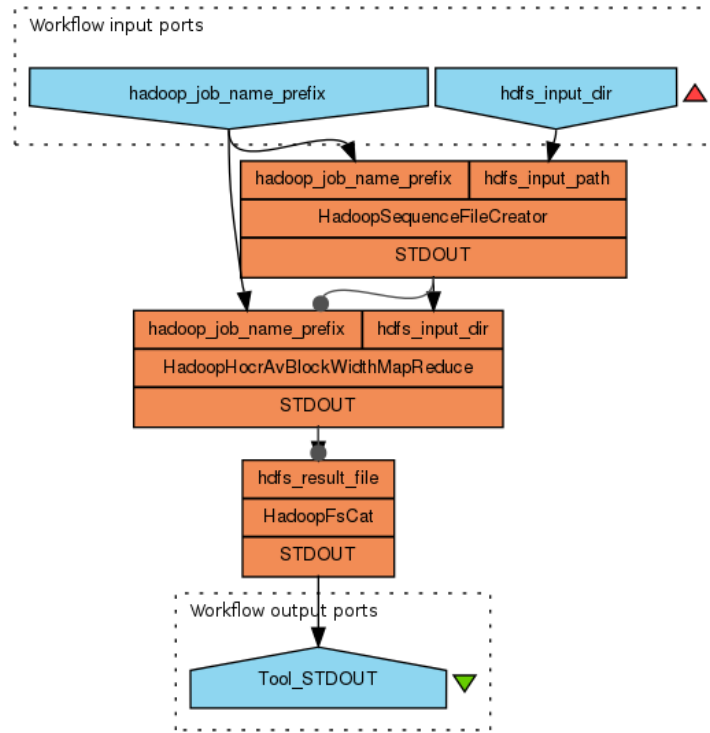


Figure 4.6.1-1: Taverna Workflow chaining various components for data preparation, average block width comparison, <http://www.myexperiment.org/workflows/3069>

First of all, dealing with lots of HTML files, means that Hadoop’s “Small Files Problem” plays a role here⁵¹. In brief, this is to say that the files to process are too small for taking them directly as input for the Map function. In fact, loading 1000 HTML files into HDFS in order to parse them in a Map function would let the Hadoop JobTracker create 1000 Map tasks. Given the task creation overhead, this would result in a very bad processing performance. In short, Hadoop does not like small files, on the contrary, the larger the better.

One approach to overcome this shortcoming is to create one large file, a so called *SequenceFile*⁵², in a first step, and subsequently load that into HDFS. These two steps are handled by the *HadoopSequenceFileCreator* Taverna component in the figure above. The component is based on a Map function which reads HTML files directly from the file server, and stores a file identifier as ‘key’ and the content as *BytesWritable* ‘value’ (key-value-pair), as illustrated in figure 4.6.1-2:

⁵¹ <http://www.cloudera.com/blog/2009/02/the-small-files-problem/>

⁵² <http://hadoop.apache.org/common/docs/current/api/org/apache/hadoop/io/SequenceFile.html>

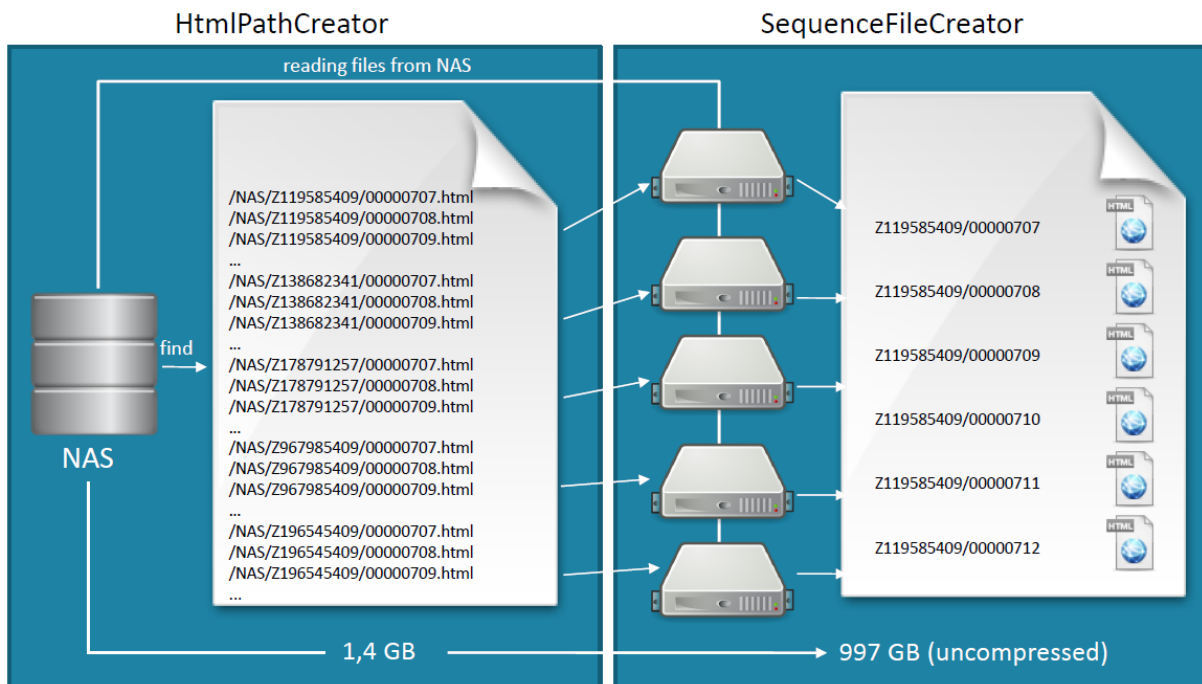


Figure 4.6.1-2: Illustration of the SequenceFile creation process.

As each processing node of the cluster has access to the file server, and given that each node executes several tasks simultaneously using all CPU cores of the worker nodes, the *SequenceFile* is created in a parallelised manner, limited by the bandwidth of the internal network (in this case *SequenceFile* creation is highly I/O bound). Using block compression for the sequence files, there will be less I/O traffic when running Hadoop jobs later on.

The *JobTracker* can then split the *SequenceFile* into 64MB splits, so that each *TaskTracker* parses a bundle of HTML files and the task creation does not weigh so much compared to the amount of data it processes.

Once the data is loaded into HDFS, the *SequenceFileInputFormat* can be used as input in the subsequent *MapReduce* job which parses the HTML files using the Java HTML parser Jsoup⁵³ in the Map function and calculates the average block width in the Reduce function. This is done by the *HadoopHocrAvBlockWidthMapReduce* Taverna component.

⁵³ <http://jsoup.org>

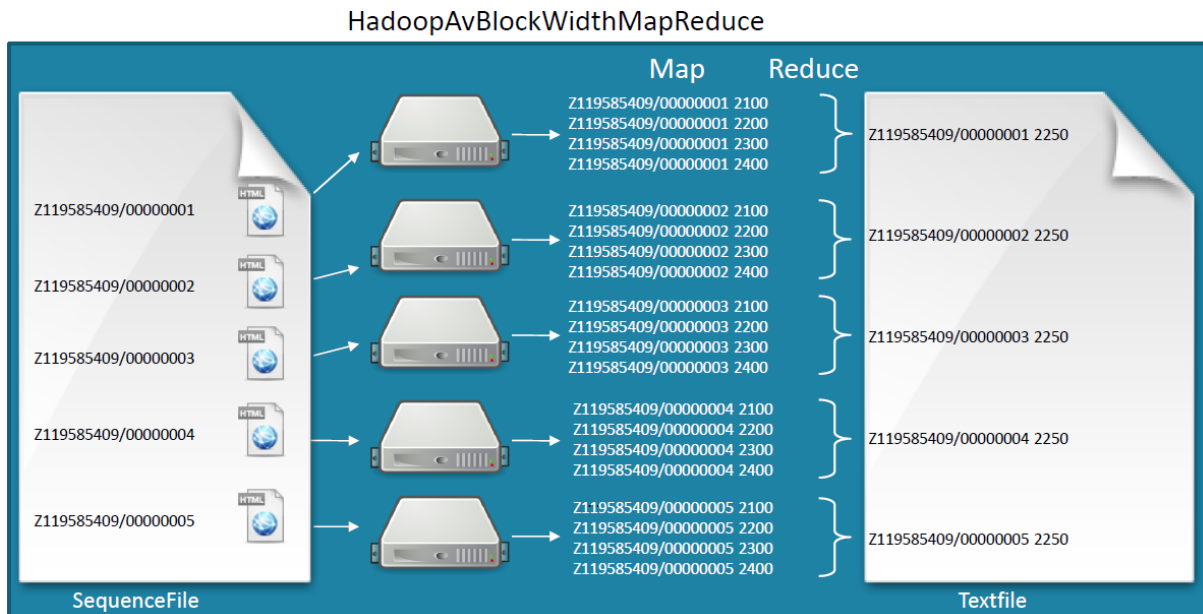


Figure 4.6.1-3: Illustration of the average block width calculation Hadoop job.

In the Taverna workflow, the handover mechanism between two different Hadoop jobs is simply established by the first job writing the output HDFS path to standard out, which the second job then takes as the HDFS input path. The second job only starts after the first job has completed.

Figure 4.6.1-3 illustrates the MapReduce job of the workflow. To explain the MapReduce job, let k_1 , as the identifier of the HTML file (data type: `org.apache.hadoop.io.Text`), and v_1 , as the value holding the content of the HTML file (data type: `org.apache.hadoop.io.BytesWritable`) be the key-value pair $\langle k_1, v_1 \rangle$ input of the Map function. A book page usually contains several block level elements, therefore the Mapper writes one $\langle k_1, v_1 \rangle$ key value pair for each block that the parser finds. The value is a string with coordinates, width, and height of the block element.

The Reduce function now receives a $\langle k_1, \langle v_1 \rangle \rangle$ list input, so that we can iterate over the blocks $\langle v_1 \rangle$ of each HTML file k_1 in order to calculate the average block width. The output of the Reduce function is then $\langle k_1, v_2 \rangle$, v_2 (data type: `org.apache.hadoop.io.LongWritable`) being the average block width.

Finally, the *HadoopFsCat* Taverna component simply writes content of the result file out to standard out which is only used for demonstration on small data sets.

Job execution can be monitored in the Taverna Workflow Workbench as shown in figure 4.6.1-4 (if the component is grey, processing finished successfully).

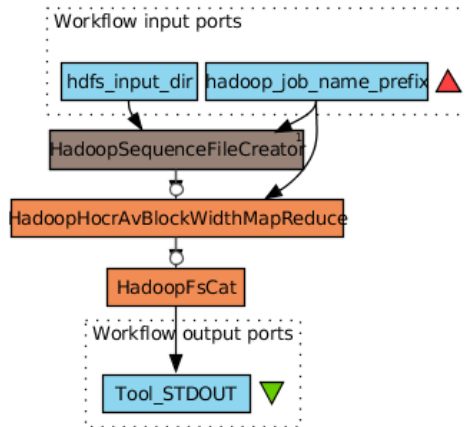


Figure 4.6.1-4: Average block width calculation Hadoop Job execution view in Taverna.

For long running jobs it makes sense to consult the web based Hadoop MapReduce Administration web-interface. A screenshot of the running job Hadoop sequence file creation job is shown in figure 4.6.1-5.

Running Jobs

Jobid	Priority	User	Name	Map % Complete	Map Total	Maps Completed	Reduce % Complete	Reduce Total	Reduces Completed
job_201208071605_0001	NORMAL	onbscs	taverna_testrun_1_seqfilecreation	9,03%	1	0	0,00%	1	0

Figure 4.6.1-5: Screenshot of the sequence file creation Hadoop job in the Hadoop Map/Reduce Administration.

The use case is extended by including image metadata (image width) of the book page images. The extended workflow includes a Hadoop Streaming API component (*HadoopStreamingExiftoolRead*) based on a bash script for reading image metadata using Exiftool, as illustrated in figure 4.6.1-6.

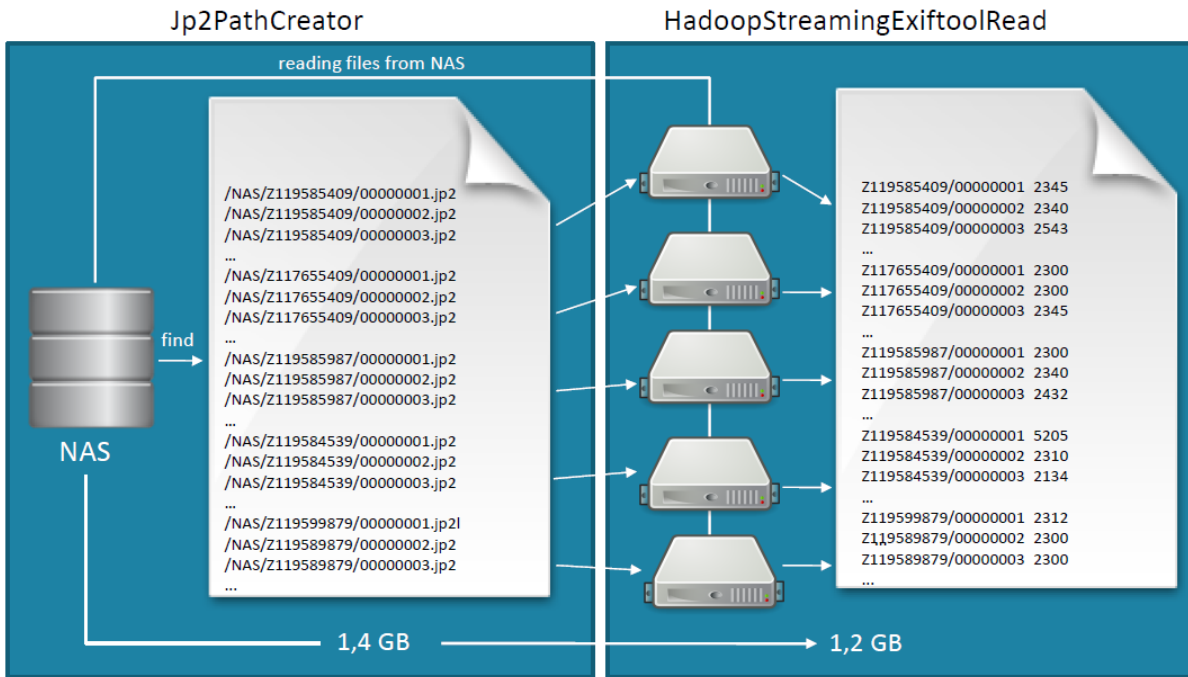


Figure 4.6.1-6: Illustration of reading the image width using Exiftool via the Hadoop streaming API.

It then uses the MapReduce component (*HadoopHocrAvBlockWidthMapReduce*) from the previous workflow. Additionally, Hive components for creating data tables and performing queries on the result files are used, as shown in figure 4.6.1-7.

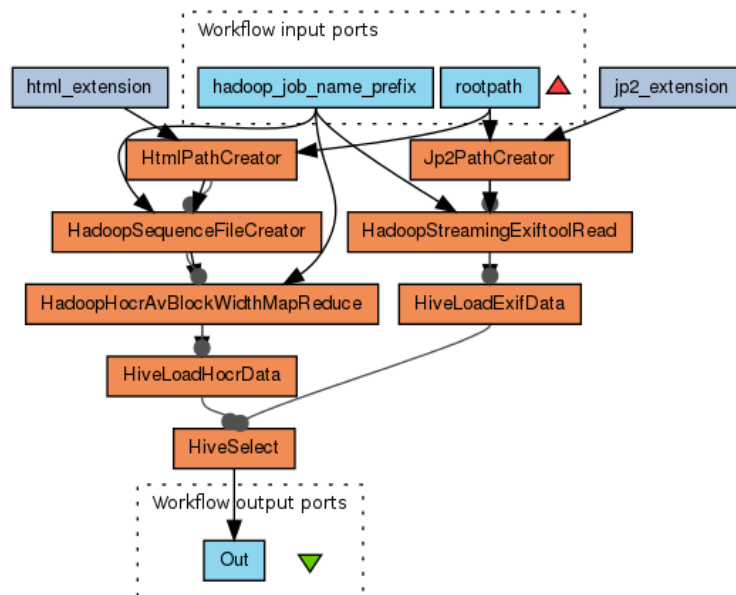


Figure 4.6.1-7: Screenshot of the Taverna workflow for using Hive.

The purpose of this workflow is to extract specific properties from the book page images and HTML data in order to make it available for analytic queries using Hive's MySQL-like query language. The

'HiveSelect' component is for testing that data has been loaded successfully and to do so it executes a SELECT query with a JOIN on the two tables created by Hive:

```
select hocr_data.identifier,hocr_data.width,exif_data.width
from hocr_data inner join exif_data on
hocr_data.identifier=exif_data.identifier;
```

An example of the results from the Hive query to compare the image width against the average block width in the HTML is shown in the following table:

Identifier	Average width	Exif width
Z119585409/00000218	1041	2210
Z119585409/00000219	826	2245
Z119585409/00000220	1122	2266
Z119585409/00000221	1092	2281
Z119585409/00000222	1026	2102
Z119585409/00000223	1046	2217
Z119585409/00000224	864	2263

During the workflow design phase, small data sets (such as one book with 815 pages) were used to study the execution performance of the components involved and analyse where improvement was needed. Figure 4.6.1-8 shows the execution log of the workflow with average execution times per component.

Name	Average time/iteration
▼ Hadoop_hOCR_parser	5,6 m
⊗ HadoopHocrAvBlockWidthMapReduce	44,9 s
⊗ HadoopSequenceFileCreator	3,3 m
⊗ HadoopStreamingExiftoolRead	2,7 m
⊗ HiveLoadExifData	31,4 s
⊗ HiveLoadHocrData	7,4 s
⊗ HiveSelect	1,3 m
⊗ html_extension - html	21 ms
⊗ HtmlPathCreator	5,4 s
⊗ jp2_extension - jp2	0 ms
⊗ Jp2PathCreator	4,8 s

Figure 4.6.1-8: Execution log of the workflow with average execution times per component.

In this case, it can be seen that the MapReduce job runs about 45 seconds and therefore it was decided to focus on this component for improving the overall workflow runtime.

To conclude, Taverna offers a simple way of linking Hadoop jobs using Taverna's "Tool" service invocation mechanism.

The principal use of the *Taverna Workbench* is for demonstrating and sharing workflows during the design and development phase. Taverna can be started from the command line⁵⁴ since Taverna version 2.3, so that it is not necessary to keep a GUI instance of the Taverna workbench accessible during the workflow runtime, but the workflow can be started as a background process instead.

```
${taverna-install-dir}/taverna-2.3.0/executeworkflow.sh -  
embedded -inputvalue rootpath ${path-to-input-dir} -inputvalue  
${job-name-prefix} -outputdir ${output-dir} ${path-to-  
workflow}/Hadoop_hOCR_parser_with_exiftool.t2flow
```

4.7. User Story: Repository Profiling

This user story is:

As a memory institution I would very much like to ensure that I'm not the only institution holding specific file formats - spreading the risk in case of lack of migration-tools etc.

The context is that many repositories have similar content and are facing similar issues, but don't have the means to share what they have and discover synergies in an easy way, beyond informal community interaction. They want to be able to share and discover factual, reliable information, no matter which repository is in use. Specifically they want to:

- be able to discover who else is holding content of a specific type (file format ID)
- know whether they are the only ones using a specific tool
- know the answer to questions like "How many uses Fedora Commons version 3.4 ?"

The SCAPE solution is SCOUT⁵⁵, a preservation watch system being developed within the SCAPE project. Characterisation components such as FITS⁵⁶ and Apache Tika⁵⁷ are used to generate repository profiles to share with SCOUT. A variety of organisations both internal and external to the SCAPE project could be connected to SCOUT to share information.

Using SCOUT allows organisations to:

- a. Harness the wisdom of the crowds
- b. Find and share expertise
- c. Create community synergy
- d. Discover opportunities
- e. Common management of risks
- f. Enhance reputation (top contributors on the front page of SCOUT)
- g. Benefit from the experience of others
- h. Use SCOUT as a preservation guide

These are the four steps to get engaged

1. Access SCOUT - get access to the community knowledge

⁵⁴ <http://www.taverna.org.uk/documentation/taverna-2-x/command-line-tool/2-3/>

⁵⁵ <http://openplanets.github.io/scout/>

⁵⁶ <http://projects.iq.harvard.edu/fits>

⁵⁷ <http://tika.apache.org/>

2. Create content profile - get information about your collection / find out what you have
3. Share content profile - join the community and discover synergies
4. Define your interests - get notified about opportunities and risks, e.g. "who else is..." & "am I the last one who..."

The User Requirements are:

I need a watch/monitor tool that, based on collection profiles for as many repositories as possible, can and will tell me when the number of repositories holding a specific format (ID) falls below a defined threshold

- SCOUT is the tool that SCAPE has developed to do this job
- A critical element is to have as many repositories as possible connected to SCOUT

Issues hampering an experiment:

- We need to consider how we can connect with SCOUT, and whether there is one central instance or many separate instances
- We require access to a repository that can be profiled (i.e. access to one or more production repositories)

No experiments have been created for this user story, or are expected to be. Development of SCOUT adaptors have come too late to incorporate tests of SCOUT within testbed experiments. It is envisaged that it will be taken up subsequently by the content holders, who wish to connect to a centrally hosted SCOUT instance.

4.8. User Story: Validation of Archival Content Against an Institutional Policy

The user story is:

As a memory institution, I want content in our repositories to conform to the corresponding file format specification, and the file format profile to conform to our institutional policies; so that our content, existing as well as future, always has the appropriate quality as specified by the file format specification and our institutional policies.

Furthermore, the following requirements and assumptions are present:

1. We assume that the content file format is known - i.e. we know the collection is a number of MPEG-1 movies.
2. We need to be able to validate the file against its file format specification/structure.
3. We need to be able to define the expected file format profile to be machine readable.
4. We need to be able to compare the expected file format profile with the actual file format profile.

4.8.1. Experiment: Validate JPEG2000 Newspapers Using Jpylyzer

Dataset	Danish newspaper - Morgenavisen Jyllandsposten ⁵⁸
Platform	SB Hadoop Platform ⁵⁹
Workflow(s)	This experiment is using Hadoop MapReduce jobs.

The idea behind this experiment is that you have a digital newspaper collection, in JPEG 2000⁶⁰ format, and you want to verify that certain properties hold true for every file in the collection. The properties that should hold true are specified in a control policy

The first step in the workflow will be to use Jpylyzer⁶¹ on each file in the newspaper collection for extraction of metadata. The second step will compare the extracted metadata against the control policy and report any differences. This is outlined on Figure 4.8.1-1.

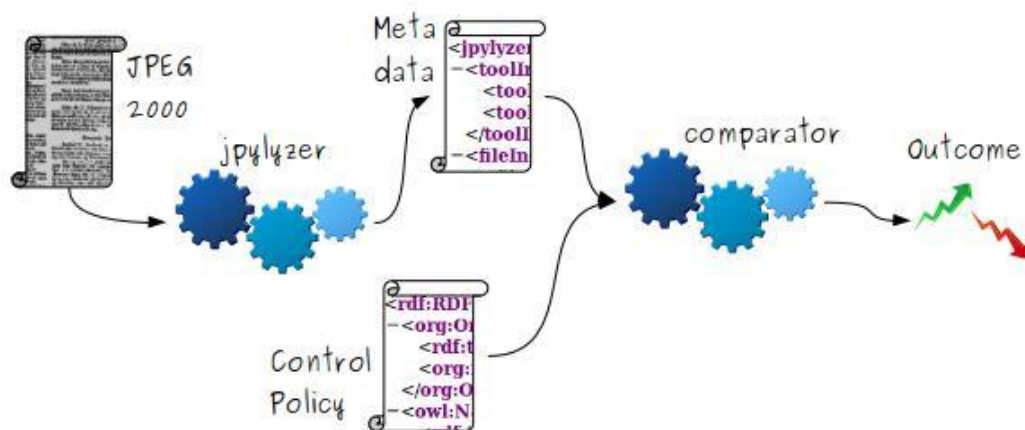


Figure 4.8.1-1: Conceptual workflow for JP2 validation

Workflow version #1

The first iteration of the experiment used a very simple setup and focus on processing the files using Jpylyzer - to get a first indication of the performance without any added complexity. Therefore, files were read from local storage instead of using the repositories as would normally be the case. Moreover, output from the processing was discarded – output from failing processes being the exception - instead of being stored in the repositories.

Workflow version #2

Building upon the results from the first version of the first iteration, the experiment was modified to better reflect an in-production workflow. The setup was further extended by adding content

⁵⁸ <http://wiki.opf-labs.org/display/SP/Danish+newspaper+-+Morgenavisen+Jyllandsposten>

⁵⁹ <http://wiki.opf-labs.org/display/SP/SB+Hadoop+Platform>

⁶⁰ http://en.wikipedia.org/wiki/JPEG_2000

⁶¹ <http://openplanets.github.io/Jpylyzer/>

repositories, where data will be read from and written to. In detail, a Fedora 3⁶²-based repository will be used for reading and writing content meta-data, and a bit repository⁶³ for reading content. By adding these systems, it is necessary to extend the experiment with components that can load and store data in an efficient manner.

5. Other Large Scale Execution Methods

Whilst the majority of work that has been undertaken within this work package has made use of the SCAPE Platform technologies, work has also been undertaken with other technologies. This work is detailed here.

5.1. Using Rosetta

Introduction

Ex Libris LSDRT experiments have been geared toward implementing SCAPE tools within the context of Rosetta, a commercial preservation system. For the purpose of SCAPE, Rosetta functionality has been expanded to support loading SCAPE objects and accept RESTful API requests by the SCAPE Loader Application. Other SCAPE tools were integrated using the existing Rosetta extendible Plugin framework.

Due to external circumstances, the only possibility for a Rosetta Testbed instance was a local environment, hosted by Ex Libris. This eliminated the possibility of testing large-scale datasets. It was decided, therefore, that expanding a Rosetta environment and confirming an expected growth in throughput should demonstrate scalability.

Experiments

The British Library provided a sample dataset of images for test purposes. The dataset consisted of approximately 1000 images in the TIFF format. These were converted into two formats – JPEG2000 and PDF – in order to test the SCAPE Jpylyzer and DRMLint tools (see below). Each of the two formats were wrapped (separately) in METS containers according to the SCAPE Data Model and loaded into Rosetta via the Loader Application.

The Loader Application communicates with two Data Connector APIs – loading an Intellectual Entity (IE) and retrieving a SIP status. This required developing a REST API layer that communicates with native Rosetta (SOAP) APIs and transforms/maps output according to the Data Connector API requirements. For example, Rosetta differentiates between a SIP ID and an IE PID (since Rosetta has a one-to-many SIP-IE relationship), while SCAPE does not. And since the Rosetta ingest workflow includes a series of validation checks (see below), synchronous creation of IEs and returning an IE PID is not possible. The Loader Application therefore calls the asynchronous ingest Data Connector API, for which Rosetta returns a SIP ID. Per API requirements, Rosetta maps this value to the IE created for the SIP (relying on the SCAPE one-to-one SIP-IE relationship), so that it can be used to request the lifecycle status of the SIP and eventually retrieve the IE itself once a PID is generated.

⁶² <http://www.fedora-commons.org/>

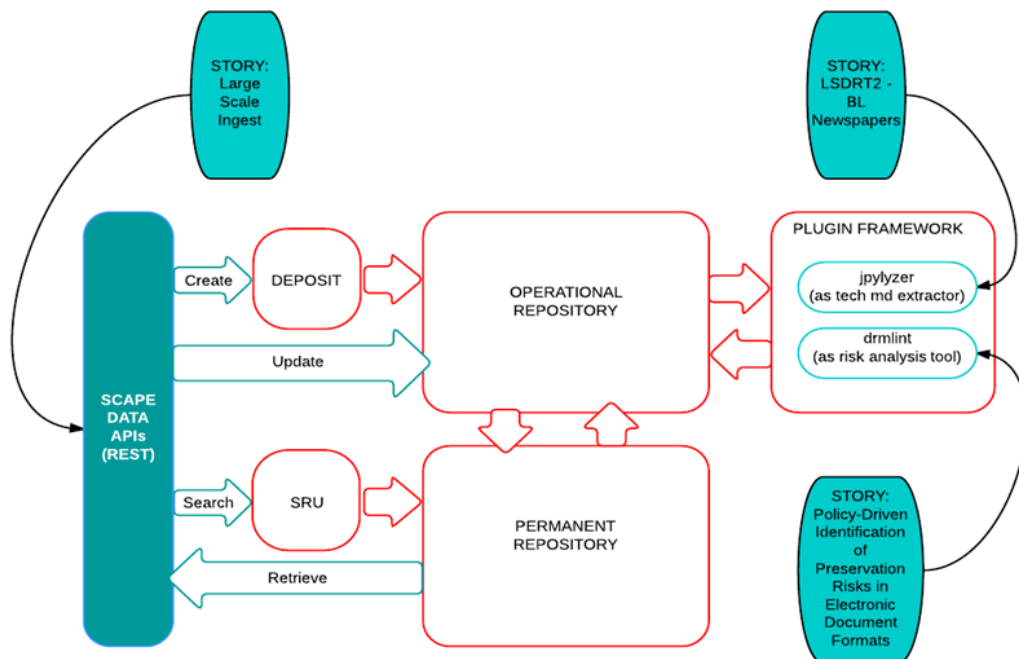
⁶³ <http://www.bitrepository.org>

The Loader Application stores a local database of SIP IDs generated during the session (multiple simultaneous processes are not supported). The Loader Application session remains alive until all METS files have been submitted to Rosetta, and subsequently requests their lifecycle status until all SIPs have completed the ingest process which can be either success, failure, or other, as defined by the Data Connector API requirements, At this point the Loader Application clears its database and terminates.

The Ex Libris LSDRT experiments therefore consist of three components: (1) Testing the Loader Application by loading SIPs that contain files to test (2) Jpylyzer, and (3) DRMLint.

The Rosetta SIP loading workflow includes a native validation stack component, which utilises a variety of standard tools to identify file formats, extract metadata, extract and/or compare checksum values, run virus checks, and identify risk factors. Each of these components is based on a plugin framework into which users can plug in industry-standard tools (DROID, JHOVE, etc.). The two experiments described here involve plugging in SCAPE tools: Jpylyzer as a metadata extractor, and DRMLint as a risk extractor.⁶⁴ Each tool was plugged into Rosetta using a standard Java wrapper interface.⁶⁵

The following diagram illustrates the integration points between SCAPE components and Rosetta:



⁶⁴ Rosetta Risk Extractors are metadata property- driven tools that register information in a designated “amd” section of the Rosetta METS object. This information is later harvested by the Rosetta Preservation component when creating a preservation set. In this experiment, the existence of rights-management properties are registered as a risk by DRMLint.

⁶⁵ See documentation in <https://developers.exlibrisgroup.com/resources/rosetta/javadoc/com/exlibris/dps/sdk/techmd/MDEXtractorPlugin.html>; <https://developers.exlibrisgroup.com/resources/rosetta/javadoc/com/exlibris/dps/sdk/risks/RiskExtractor.html>.

Defining 'Scalability': Rosetta Environment Architecture

The Rosetta environment is comprised of one or more application servers running JBoss AS and an Oracle server. Communication between the application and database during SIP processing is based on JMS queues and a worker thread pool. SIP processing can be scaled by adding workers to the pool – either from the existing application servers or, in case the load on the servers is already high, by adding to them.

LSDRT experiments were performed in an environment with three application servers, which represents an average-size Rosetta cluster.⁶⁶ During the first loading, the number of workers on one server was left at default, while two of the servers was set to 0. During a second loading all servers were set to the same (default) number of workers.⁶⁷ The expected result was that the duration of the tools' processing would not be affected by the number of workers or servers, demonstrating that these tools' performance, when used within Rosetta SIP processing, scales in arithmetic progression.

Methodology and Integration

During SIP validation, a set of rules determines which tools are relevant for the incoming file formats. The rules are based on the PRONOM unique identifier (format ID). That is to say, the tools that will be deployed by the SIP validation stack (e.g. which metadata extractor, which risk extractor) are determined by the outcome of its first stage, which is a process of format identification (and, if necessary, disambiguation).

Rosetta was configured to use Jpylyzer as a metadata extractor for format ID x-fmt/392 (JP2), and DRMLint as a risk extractor for fmt/18 (PDF v1.4⁶⁸). Initial testing confirmed the files in the dataset were correctly identified by the DROID-based Rosetta format identification process, the appropriate tools for each format were activated during the validation process, and each tool generated the expected output (metadata properties for JP2 files and a risk for PDF files).

To measure performance, native Rosetta application server logging provides the necessary information. Rosetta logging is configured to log the duration of each tool's runtime.⁶⁹ It was expected that the average duration of a single process would remain constant when running one or more application servers (while duration of the overall loading process in the latter case will naturally decrease).

Conclusion

Initial results indicate performance of the tools was not affected by adding workers, indicating that Rosetta will scale arithmetically when using Jpylyzer and DRMLint.

⁶⁶ Based on Rosetta customer usage, as of January 2014.

⁶⁷ The exact number of worker threads is determined by an internal algorithm. For the purpose of this experiment and report suffice it to say that the load on all servers was set identically.

⁶⁸ PRONOM assigns IDs fmt/14-20 to PDF versions 1.0-1.6, respectively. PDFs created for the purpose of this test were all v1.4.

⁶⁹ Duration is measured in ms when under one second, and rounded to a full second when over a second.

5.2. Using Microsoft Azure

Work has been undertaken by Microsoft Research to use Microsoft Azure for two purposes:

1. Characterisation of files using Apache Tika and DROID
2. Producing cloud services for format migration: “SCAPE Azure Services”⁷⁰⁷¹

This work will be described in deliverable D11.3.

SCAPE Azure Services⁷² provides a REST API to perform the following tasks:

- Upload / download / delete a file
- Get information about an uploaded file
- Get a list of supported file format conversions and convert files
- Compare two files

Currently the system supports DOC/DOCX, ODT, PDF and RTF, amongst other file formats. A SCAPE Azure Client Toolkit (CLIKIT) has been developed to interface with the SCAPE Azure Services and this toolkit could be used within a large-scale workflow.

No LSDRT workflows currently exist for using SCAPE Azure Services; however, benchmarking SCAPE Azure Services for migrating documents is planned.

5.3. Using Apache Pig within the Execution Platform

Work to use Apache Pig for executable workflows was undertaken within the Platform subproject. Although this work was not undertaken within this work package, it is complimentary to the work described here, as it describes a workflow for TIFF to JP2 migration. It is discussed in much greater detail in deliverable D6.3.

In short, the two approaches described there are:

1. Create an Apache Pig script as a workflow
2. Create a Taverna workflow and have it converted to an Apache Pig script (experimental)

6. Conclusion and next steps

There has been a wide variety of work undertaken within this work package, making use of, and feeding back into, other SCAPE outputs. Through the large-scale workflows developed here, the various ways in which SCAPE outputs can be used and combined with other platforms and tools has been demonstrated.

⁷⁰ <https://lib.stanford.edu/files/pasig-oct2012/14-Milic-Frayling-SCAPE-Azure-FormatConversion-PASIG%2712-FINAL2.pdf>

⁷¹ <http://wiki.opf-labs.org/display/SP/SCAPE+Azure+Platform>

⁷² <http://scapestaging.cloudapp.net:8080/>

Different methods for parallelising the same workflow have been implemented, which additionally demonstrates how standard tools and software can be parallelised. It has been demonstrated how the SCAPE Execution Platform, and specifically Apache Hadoop, can be used to parallelise standard software and Taverna workflows. In addition it was shown how Fedora 4, Rosetta and Apache Hive could be used within the SCAPE Execution Platform ecosystem.

Good links have been fostered with other sub-projects, particularly Platform and Preservation Components.

The work undertaken within this work package can be further developed by making use of the SCAPE Platform Apache Pig work described in D6.3. However, this is not something that will be worked on within this work package before the end of the SCAPE Project.

For various reasons, including tool readiness, not all of the experiments described above will have a full evaluation. Full evaluation testing is currently underway and results for experiments with complete workflows will be reported in deliverable D18.2.

7. Glossary

This glossary is derived from the *SCAPE Project Glossary* that is canonically held here: <http://wiki.opf-labs.org/display/SP/SCAPE+Glossary>

Term	Abbreviation	Definition
Action Service		An action service is a type of a digital preservation service that performs some kind of action on a digital object, e.g. migrating the object to a new file format.
Apache Hadoop		Framework for processing large data sets on a computer cluster. See http://hadoop.apache.org
Apache Pig		A high-level language for creating workflows that run on top of Hadoop/MapReduce
Apache Tika		Software for identifying file formats. See https://tika.apache.org/
Automated Planning		A systematic and semi-automatic process that provides the ability to assess the impact of influencers and specify actionable preservation plans that define concrete courses of actions and the directives governing their execution. This is the operative management of obsolescence and maximizing expected value with minimal costs.
Automated Watch		A systematic and semi-automatic process that provides the ability to monitor external and internal entities for changes having a potential impact on preservation and to provide notification. The Automated Watch Component denotes the software component that supports the Automated Watch process.

Azure Platform		A cloud-based service, providing virtualized services such as Hadoop clusters
Bitstream		A bitstream is contiguous or non-contiguous data within a file that has meaningful common properties for preservation purposes. A bitstream cannot be transformed into a standalone file without the addition of file structure (headers, and so forth) and/or reformatting to comply with a particular file format.
Characterisation Service		A characterisation service is a type of a digital preservation service that extracts any kind of information from a digital object, as an identifier or file related properties, for example.
Cloud		Environments which provide resources and services to the user in a highly available and quality-assured fashion, thereby keeping the total cost for usage and administration minimal and adjusted to the actual level of consumption.
Cloud Computing		A pay-per-use model for enabling available, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, services) that can be rapidly provisioned and released with minimal management effort or service-provider interaction.
(SCAPE) Components		SCAPE components are <i>Taverna Components</i> , identified by the SCAPE Preservation Components sub-project, that conform to the general SCAPE requirements for having annotation of their behaviour, inputs and outputs. SCAPE components may be stored in the <i>SCAPE Component Catalogue</i> .
(SCAPE) Component Catalogue		The Component Catalogue is a searchable repository for the definitions of <i>SCAPE Components</i> , Component Families and Component Profiles. The component catalogue is implemented by the myExperiment service and implements the Component Service API .
Component Lookup API		[Part of the Component API]
Component Management		Tools and the Component Catalogue Service encompassing the creation, storage and cross-organisational sharing of SCAPE Components.
Component Profile		A definition of an interface that a Component should conform to. A Component profile defines what input ports and output ports the Component must have, what inputs and outputs may be optionally present, and what semantic annotations may be attributed to the Component and its ports.
Component Registration API		[Part of the Component API] A REST API to be implemented by Digital Object Repositories to allow

		SCAPE components to access the content and preservation plans held on the repository.
Control Policies		Policies that formulate the requirements for a specific collection, a specific preservation action, for a specific designated community This level can be human readable, but should also be machine readable and thus available for use in automated planning and watch tools to ensure that preservation actions and workflows chosen meet the specific requirements identified for that digital collection.
Data locality		“Data locality” refers to the fact that Hadoop tries to assign map tasks to nodes that are close to the data, i.e. the processing cores are on the same machine as the hard disk storing the data blocks.
Data Publication Platform	DPP	A platform supporting the publication of data sets, e.g. experimental SCAPE data, as Open Linked Data.
Digital Object Repository	DOR	An OAIS Compliant repository that provides a data management solution for storing content and metadata about digital objects, as well as Preservation Plans. DORs implement three interfaces: Plan Management API; Data Connector API; and the Report API.
DROID		Software developed by the National Archives (UK) to determine a unique file format identifier (PUID, see corresponding glossary entry). DROID is a software tool developed by The National Archives (UK) to perform identification of file formats. See http://digital-preservation.github.io/droid/
Execution Environment		An abstract layer of the Execution Platform which provides a placeholder representing functionality to be fulfilled by a specific technology. The Execution Environment provides the physical infrastructure to perform computation. An example might be the nodes of a Hadoop cluster.
Execution Platform	EP	An infrastructure that provides the computational resources to enact a Preservation workflow and execute Preservation actions. Abstracted into three layers: the Execution Environment; the Job Execution Service and the Job Submission Service API. It could otherwise be described as an extensible infrastructure for the execution of digital preservation processes on large volumes of data (using a combination of Apache Hadoop and Taverna)
(SCAPE) Experiment Evaluation		Findings and results, both measurable and non-measurable, of a particular execution of an Experiment, within the Testbed sub-package.
(SCAPE) Experiment		A unit of work that defines an implementation of a User Story, within the Testbed sub-package. It consists of a

		dataset, one or more preservation components, a workflow and a processing platform that can be used to evaluate SCAPE technology and provide evidence of scalable processing
FFprobe	FFprobe	FFprobe is a tool that belongs to the FFmpeg family and is used to gather information about multi-media-files. http://ffmpeg.org/ffprobe.html
File Format Characterisation		The process of determining the properties of a file format, for example, the bit depth, colour space, width of an image, the frames per second of a video, etc.
File Format Identification		The process of determining the identity of a file format instance, typically by assigning an identifier, as the PUID (see corresponding glossary entry) as a precise identifier or a MIME Type (see corresponding glossary entry) identifier as a vague file type identifier.
Hadoop		See Apache Hadoop.
HDFS	HDFS	Hadoop Distributed File System. This is Hadoop's file system which is designed to store files across machines in a large cluster.
HBase	HBase	Distributed database on top of Hadoop/HDFS, see https://hbase.apache.org
Intellectual Entity	IE	A set of content that is considered a single intellectual unit for purposes of management and description – for example, a particular book, map, photograph, or database. An intellectual entity may have one or more digital representations.
Job Execution Service	JES	An abstract layer of the Execution Platform which provides a placeholder representing functionality to be fulfilled by a specific technology. The Job Execution Service provides job scheduling functionality, allocating computing tasks amongst the available hardware resources available within the Execution Environment. An example might be Taverna-Server or Hadoop.
Job Submission Service	JSS	An abstract layer of the Execution Platform which provides a placeholder representing functionality to be fulfilled by a specific technology. Provides the entry point to the Execution Platform, implementing a remotely accessible interface to enable a user or client application to schedule and execute workflows (jobs) on the Execution Environment. The exact interface depends on the underlying Job Execution Service and Execution Platform, but typical examples would be the Hadoop API provided over a SSH connection, or the Taverna-Server REST API over HTTP.
Loader Application		A component that loads Digital Objects into a Digital Object Repository that implements the SCAPE Data Connector API.

Map/Reduce	MR	A programming paradigm for processing large data sets using a parallel, distributed algorithm on a Hadoop cluster.
Microsoft Azure Platform		See Azure Platform
MIME Type		A standard identifier used on the Internet to indicate the type of data that a file contains.
MyExperiment		A web application to allow users to find, use and share scientific workflows and other Research Objects, and to build communities around them.
NFS		Network File System
Plan Management API		An API to be implemented by Digital Object Repositories that provides HTTP endpoints for the retrieval and management of Preservation Plans.
Plan Management Service	PMS	Any service that implements the Plan Management API is a Plan Management Service. Note that a PMS may also implement other APIs and be <i>principally</i> known by other names.
Plato		A web-based tool that creates a <i>Preservation Plan</i> and provides a user interface for viewing, managing and updating that plan. The plan itself is stored in the <i>Plan Management Service</i> after creation.
Preservation Component	PC	See SCAPE Component
Preservation Plan		A preservation plan is a live document that defines a series of preservation actions to be taken by a responsible institution due to an identified risk for a set of digital objects or records (called a collection). It is defined by <i>Plato</i> and stored in a <i>Plan Management Service</i> .
Program for parallel Preservation Load	PPL	An application that takes an existing Taverna Workflow as an input and automatically generates a Java class file that can be executed on a Hadoop cluster.
PRONOM		PRONOM is an information system about data file formats and their supporting software products. See https://www.nationalarchives.gov.uk/PRONOM
Pronom Unique Identifier	PUID	The PRONOM Persistent Unique Identifier (PUID) is an extensible scheme for providing persistent, unique and unambiguous identifiers for records in the PRONOM registry. Such identifiers are fundamental to the exchange and management of digital objects, by allowing human or automated user agents to unambiguously identify, and share that identification of, the representation information required to support access to an object. This is a virtue both of the inherent uniqueness of the identifier, and of its binding to a definitive description of the representation information in a

		registry such as PRONOM. From: http://www.nationalarchives.gov.uk/aboutapps/pronom/puid.htm
Pronom Signature File		Signature files are generated by PRONOM (see corresponding glossary entry) and used by DROID (see corresponding glossary entry) for file format identification. The signature file contains a subset of the information from the PRONOM knowledge base required by the DROID software to perform the file format identification. See https://www.nationalarchives.gov.uk/aboutapps/pronom/droid-signature-files.htm
Preservation Watch		See Automated Watch
Quality Assurance Component		A Quality Assurance Component is used to determine a quality measure related to the outcome of applying an Action Service (see corresponding glossary entry) to a digital object.
Results Evaluation Framework	REF	A generic semantic system for evaluating large datasets of experimentation results in a simple fashion
Report API		An OAI_PMH based API to be implemented by Digital Object Repositories that enables the SCAPE Automated Watch component to retrieve information about the state of the repository.
Rosetta Platform		A digital preservation repository/system produced by Ex Libris
Scalable Preservation Environments	SCAPE	An EU funded project developing scalable services for the planning and execution of institutional preservation strategies on an open source platform that orchestrates semi-automated workflows for large-scale, heterogeneous collections of complex digital objects.
SCAPE Characterisation Component		Characterisation components are a family of <i>SCAPE Components</i> (defined to wrap tools produced in WP9) that compute one or more properties of a <i>single</i> instantiated digital object or file. The output ports that produce measures are always annotated with the metric (in the <i>SCAPE Ontology</i>) that describes what the component computes.
SCAPE Migration Component		Migration components are a family of <i>SCAPE Components</i> (defined to wrap tools produced in WP10) that apply a transformation to an instantiated digital object or file to produce a new file. The input is annotated with a term (from the <i>SCAPE Ontology</i>) that says what sort of digital object/file is accepted, and the output is annotated with a term that says what sort of file is produced.
SCAPE Ontology		The SCAPE Ontology is an OWL ontology that formally

		defines the terms used by computing systems in SCAPE.
SCAPE Platform		See Execution Platform
SCAPE QA Component		QA components are a family of <i>SCAPE Components</i> (defined to wrap tools produced in WP11) that compute a comparison between <i>two</i> instantiated digital objects or two files. They produce at least one output that has a measure of similarity between the inputs, and that output is annotated with the metric (in the <i>SCAPE Ontology</i>) that describes the nature of the similarity metric.
SCAPE Story		A short and succinct high-level statement of the preservation issue encountered by a partner institution.
SCAPE Utility Component		Utility components are a family of <i>Taverna Components</i> that provide miscellaneous capabilities required for constructing SCAPE workflows, but which are not a core feature of the SCAPE preservation planning process. For example, they can provide assembly and manipulation of XML documents that contain collections of measures of workflows. Note that utility components are not SCAPE components per se; they do not conform to the standard profiles. Instead, they are used in support roles.
Scout		An Automated Watch system that provides an ontological knowledge base to centralize all necessary information to detect preservation risks and opportunities
Taverna Components		Taverna components are <i>Taverna workflow</i> fragments that are stored independently of the workflows that they are used in, and that are semantically annotated with information about what the behaviour of the workflow fragment is. They are logically related to a programming language shared library, though the mechanisms involved differ. Taverna components are stored in a component repository. This can either be a local directory, or a remote service that supports the Taverna Component API (e.g., the <i>SCAPE Component Catalogue</i>). Only components that are stored in a publicly accessible service can be used by a <i>Taverna workflow</i> that has been sent to a system that was not originally used to create it.
Taverna Command Line Tool		The Taverna Command Line Tool can execute a Taverna Workflow in a terminal/command prompt, without displaying a Graphical User Interface (GUI)
Taverna Server	TAVSERV	Taverna Server is a multi-user service that can execute <i>Taverna workflows</i> . Clients do not need to understand

		those workflows in order to execute them.
Taverna Workbench		The Taverna Workbench is a desktop application for creating, editing and executing <i>Taverna workflows</i> .
Taverna Workflow		A Taverna workflow is a parallel data-processing program that can be executed by <i>Taverna Workbench</i> or <i>Taverna Server</i> . It is stored as an XML file, and has a graphical rendering.
Tool-to-MapReduce Wrapper	ToMaR	A SCAPE developed tool which wraps command line tasks for parallel execution as Hadoop MapReduce jobs
Toolspec		An XML file written to a standard API that contains details of how to execute a tool for a particular purpose; for example txt2pdf might define how to use a command line tool to convert text to pdf. Toolspecs can have different types such as migration or QA.
Toolwrapper		The toolwrapper is a Java tool developed in the SCAPE Project to simplify the execution of the following tasks: Tool description (through the toolspec); Tool invocation (simplified) through command-line wrapping; Artifacts generation (associated to a tool invocation, e.g., Taverna workflow); and Packaging of all the generated artifacts for easier distribution and installation
(SCAPE) User Story		See SCAPE Story