




# Quality Assurance Workflow, Release 3 + Release Report

## Authors

Zeynep Pehlivan (Université Pierre et Marie Curie), Bolette A. Jurik (State & University Library Denmark), Roman Graf (AIT Austrian Institute of Technology), Willima Palmer (British Library), Johan van der Knijff (National Library of the Netherlands), Arthur Kulmukhametov (Vienna University of Technology), Leila Medjkoune, Stanislav Barton (Internet Memory)

September 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**

Whenever digital content is acquired, created, moved, unpackaged, processed, migrated, curated, repackaged or otherwise changed, problems can occur and collection damage can result. Thus, large-scale quality assurance is an essential aspect of preservation methodology. In the past, large-scale quality assurance has relied on a combination of human intervention and statistical methods. In order to meet the challenges posed by large and heterogeneous digital collections, the objective of the PC.WP3 is to provide automated, scalable methods for quality assurance, applicable to a range of preservation workflows that are aimed at ensuring long term access to digital content.

This document reports on the task of identifying, developing and testing Quality Assurance (QA) tools to be used in the SCAPE project. The report presents the work in the context of QA for different media types: Audio, Images, Web, Document Collections, Documents. For each media type, this report contains the overall description of tools, deployment guides for the related tools, scalability work, benchmarks and demonstrations.

# Table of Contents

Executive Summary .....	3
1 Introduction.....	7
2 Audio QA .....	9
2.1 Introduction.....	9
2.2 Tools and Interfaces .....	9
2.2.1 xcorrSound .....	9
2.2.2 Scalability .....	10
2.3 Taverna Workflows .....	11
2.4 Demos.....	12
2.5 Benchmarking.....	12
2.5.1 Dataset.....	13
2.5.2 Experiments and Results.....	14
2.6 Conclusion .....	16
3 Document Collections QA .....	17
3.1 Introduction.....	17
3.2 Tools and Interfaces .....	19
3.2.1 Scalability .....	21
3.3 Taverna Workflows .....	22
3.4 Demos.....	24
3.5 Benchmarking.....	24
3.5.1 Dataset.....	26
3.5.2 Experiments and Results.....	27
3.6 Conclusion .....	28
4 Web QA .....	29
4.1 Introduction.....	29

4.2	Tools and Interfaces .....	29
4.2.1	Pagelyzer .....	30
4.2.2	Scalability .....	32
4.3	Taverna Workflows .....	33
4.4	Demos.....	34
4.5	Benchmarking.....	34
4.5.1	Dataset.....	34
4.5.2	Experiments and Results.....	35
4.6	Conclusion .....	37
<b>5</b>	<b>Image QA.....</b>	<b>38</b>
5.1	Jpylyzer .....	38
5.1.1	Functionality and scope .....	38
5.1.2	Use and deployment.....	39
5.1.3	Workflows.....	39
5.1.4	Demos .....	42
5.1.5	Benchmarking .....	42
5.2	Photohawk .....	47
5.2.1	Deployment.....	48
5.2.2	Workflows.....	48
5.2.3	Experimentation .....	48
<b>6</b>	<b>QA Supporting Tools.....</b>	<b>53</b>
6.1	DRMLint.....	53
6.1.1	Exemplar Use Case.....	53
6.1.2	Tool Development.....	53
6.1.3	Testing at scale / Use in testbeds .....	54
6.2	Dissimilar .....	55
6.2.1	Exemplar Use Case.....	55
6.2.2	Tool Development.....	55
6.2.3	Testing at scale / Use in testbeds .....	56

6.3 JP2Check library ..... 57

7 Conclusion ..... 59

Appendix A: Installation of Demo Web Sites ..... 60

## 1 Introduction

The final objective of PC.WP.3 is to develop and assess methods for quality assurance that are as much as possible automated and scalable in terms of quantity of data and machines that can be used. To do so the work in the work package is not only to build the tools, but also to identify a range of preservation workflows that are aimed at ensuring long term access to digital content.

Instead of following the task descriptions, PC.WP3 has been working based on media type. The activities in the work package are carried out side by side with the other subprojects, i.e. with PW we share the common aim to define an interface to deliver QA measures to the planning and the watch components for assessment and successive reaction and feedbacks.

In order to meet the challenges posed by large and heterogeneous digital collections, automated quality assurance approaches were developed. This involved designing tools and methods to measure different properties of digital objects. Furthermore, comparison of digital objects often requires transforming digital objects into a common intermediary format that can be used for a specific analysis. The tools developed are listed by media type in Table 1:

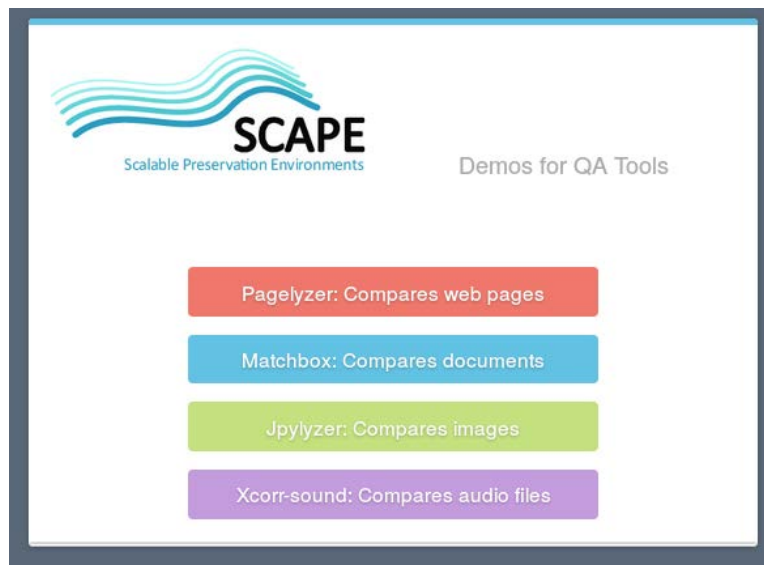
Media Type	Tool Name
Audio	xcorrSound
Documents	MS QA Tools
Document Collections	Matchbox
Web Pages	Pagelyzer
Images	Jpylyzer
Images	PhotoHawk
Images	Dissimilar
Images	JP2Check
PDFs / EPubs	DRMLint

Table 1 Overview of QA Tools

For each tool, (except some of QA Supporting tools), PC.WP3 provides deb packages, deployment guides, Taverna workflows, benchmarking results (correctness based detailed in D11.2<sup>1</sup>). Please refer to D11.2 for detailed use cases. This deliverable does not contain the work done for Documents processing by Microsoft Research due to some legal issues. One of the Optical character recognition (OCR) tools used to compare documents is not licensed for use in a cloud environment and MSR legal team is working out the details. The details of Microsoft Research work can be found in D11.2.

For 4 of our tools (xcorrSound, Matchbox, Pagelyzer and Jpylyzer), online demonstrations are also provided:

<http://scape.opf-labs.org/>



**Figure 1 Demo page for QA tools**

Appendix A describes how to set up a local VM that runs the SCAPE demonstrator for all tools by using Virtual Box and Vagrant.

---

<sup>1</sup> [https://portal.ait.ac.at/sites/Scape/Shared%20Documents/Deliverables/Final/SCAPE\\_D11.2\\_UPMC\\_V1.0.pdf](https://portal.ait.ac.at/sites/Scape/Shared%20Documents/Deliverables/Final/SCAPE_D11.2_UPMC_V1.0.pdf)

## 2 Audio QA

### 2.1 Introduction

The Audio QA work was motivated by the user story on Large Scale Audio Migration<sup>2</sup>, which describes a wish to migrate from mp3 files to wav files - large scale. The reason for this wish is homogeneity in the collection and robustness of the target format. The reason SCAPE is working on this user story is that it is a good example of an audio migration. The user requirements specify

1. There is a need for a measure of similarity between two audio files based on how they 'sound' to ensure that the migration went well. Manual quality assurance (listening to the migrated audio files) is not feasible resource wise due to the large size of collections.
2. There is a need to be able to compare properties of mp3s such as duration with those of the migrated WAV to ensure that the metadata is preserved in the migration.
3. There is a need to be able to migrate mp3 to wav.

Existing tools which meet requirements 2 and 3 are available for use, and which scale. For requirement 1 a new tool was developed, and with the new Taverna Hadoop workflow introduced below for this tool, it can be demonstrated that this tool scales as well.

### 2.2 Tools and Interfaces

The tools used in the *mp3 to wav migration and QA experiment* are *FFmpeg*<sup>3</sup> for migration, *mpg321*<sup>4</sup> for conversion, *Ffprobe*<sup>5</sup> for property extraction and *xcorrSound*<sup>6</sup> *waveform-compare* for content comparison.

#### 2.2.1 xcorrSound

The xcorrSound tool suite developed in SCAPE contains four tools:

- overlap-analysis detects overlaps in two audio files
- sound-match finds occurrences of a smaller audio file within a larger audio file or collection
- waveform-compare compares two audio files and outputs the similarity
- sound-index builds an index for sound-match to work within (in development)

The focus in this work was on the waveform-compare tool for comparing content of audio files. The tool is described in *Deliverable D11.2 Quality Assurance Workflow, Release 2 + Release Report*<sup>7</sup>. To

---

<sup>2</sup> <http://wiki.opf-labs.org/display/SP/Large+Scale+Audio+Migration>

<sup>3</sup> <http://www.ffmpeg.org/>

<sup>4</sup> <http://mpg321.sourceforge.net/>

<sup>5</sup> <http://www.ffmpeg.org/ffprobe.html>

<sup>6</sup> <http://www.openplanets.github.io/scape-xcorr-sound/>



read more about the full xcorrSound tool box, go to *the xcorrSound landing page*:

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

waveform-compare is a command line tool and can be incorporated in diverse workflows. It is written in C++, which was done mainly for performance. The Hadoop job is written in Java and uses the command line version of waveform-compare.

Deployment guides can be found:

<http://openplanets.github.io/scape-xcorr-sound/#installation-guide>

### 2.2.2 Scalability

As part of the audio QA work, an experiment has been developed and is available on Github<sup>8</sup>. The experiment currently contains four Hadoop map-reduce jobs<sup>9</sup>.

- The first migrates mp3s to wav files using the command line tool *FFmpeg*.
- The second converts mp3s to wav files using the command line tool *mpg321*.
- The third compares the content of the two wave files using the xcorrSound *waveform-compare* command line tool.
- The fourth is still in development. It extracts properties from the mp3 file and from the migrated wav file using *Ffprobe*, and it compares the extracted properties.

Common for the four jobs is that they do not use the map-reduce programming model, they all simply call an external command line tool. Using Hadoop in this fashion is simple parallelization. What is gained from Hadoop here is the easy set-up, stability and reliability.

#### 2.2.2.1 Input/Output HDFS/NFS

The file containing the list of mp3 files to be migrated is made available from HDFS. The mp3 files are stored on NFS and the resulting wav files are written to NFS. There are a number of reasons for this.

- The first is that the audio tools, that are being used, were written to read from and write to NFS.
- Also at SB digitally preserved material does not reside on HDFS, which means that in order to migrate from and to HDFS, it would be necessary to first to copy the mp3s to HDFS and later copy the wavs from HDFS. These extra copy operations are expensive, when considering large-scale audio collections.

---

<sup>7</sup> <http://www.scape-project.eu/deliverable/d11-2-quality-assurance-workflow-release-2-release-report>

<sup>8</sup> <https://github.com/statsbiblioteket/scape-audio-qa-experiments>

<sup>9</sup> <http://wiki.opf-labs.org/display/SP/Large+Scale+Audio+Migration>

- Finally the SB Hadoop Platform is set up using network storage as local storage, which means that HDFS locality property is not exploited, and thus accessing the files on NFS rather than HDFS does not present a large overhead.

The preservation event and log files are all written to HDFS. This means there is a rather complex input/output model with input from both HDFS and NFS and also with output to both HDFS and NFS. If this workflow is going to be used in production, a repository connection would have to be added, such that data can be both retrieved from the repository and written to the repository.

### ***2.2.2.2 Input/Output key-value-pairs***

When setting up the Hadoop jobs, the input/output key-value-pairs are not given.

- The input to the map function of FfmpegMigrationMapper is a line number as key (not used) and a Text line, which is the path to an mp3 file. The output is an exit code (not used), and the path to an output file.
- The input to the map function of Mpg321ConversionMapper is a line number as key (not used) and a Text line, which is the path to an mp3 file. The output is an exit code (not used), and the path to an output directory.
- The input to the map function of WaveformCompareMapper is a line number as key (not used) and a Text line, which is two tab-separated paths to two wav files to be compared. The output is an exit code, and the output of the waveform-compare tool (used to be path to log file).
- The input to the map function of FfprobeExtractCompareMapper is a line number as key (not used) and a Text line, which is a path to an original mp3 and a migrated wav file (tab-separated). The output is an exit code, and the output of the comparison.

This works well when a Taverna workflow is used to control the input and output to the different Hadoop jobs. However, this is not optimal by using Chainmapper to combine the different mappers. An interesting performance comparison would be to compare a Chainmapper workflow with an equivalent Taverna workflow.

## **2.3 Taverna Workflows**

The Taverna workflow for Combined Migration, Validation, Feature Extraction, and Comparison presented in D16.1 LSDR Executable Workflows for Experimental Execution<sup>10</sup> showed the full workflow for mp3 to wav audio migration including QA:

---

<sup>10</sup> <http://www.scape-project.eu/deliverable/d16-1-lsdr-executable-workflows-for-experimental-execution>

- Migrate Mp3 files to Wav files
- Perform QA
  - File Format Validation
  - Significant Property Comparison
  - Content comparison

This workflow is available from My Experiment<sup>11</sup> in a test version. The new Taverna workflow<sup>12</sup> uses the Hadoop jobs for the different steps. The current “slim” version only does migration and content comparison combining three of the Hadoop jobs. This workflow does migration, conversion and content comparison. The top left box (nested workflow) migrates a list of mp3s to wav files with a Hadoop map-reduce job which uses the command line tool FFmpeg, and outputs a list of migrated wav files.

The top right box converts the same list of mp3s to wav files using another Hadoop map-reduce job which uses the command line tool mpg321, and outputs a list of converted wav files. This conversion “plays” the original mp3 file into waveform or more accurately into the wav file format, which the waveform-compare tool uses to compare the original sound waves with the migrated sound waves.

The Taverna work flow puts the two lists of wav files together and the bottom box receives a list of pairs of wav files to compare. The bottom box compares the content of the paired files using a Hadoop map-reduce job which uses the xcorrSound waveform-compare command line tool, and outputs the results of the comparisons. Note that the nested workflows can also be used independently.

## 2.4 Demos

The xcorrSound tool box has an online *demo site*: <http://scape.opf-labs.org/xcorr-sound/>. The current version only demonstrates *waveform-compare* on uploaded files. A new version has been developed that demonstrates three xcorrSound tools and also offers example files for the demonstrations. This version is not online yet.

## 2.5 Benchmarking

The experiments are performed on the SB Hadoop Cluster<sup>13</sup>. This cluster was not set up on commodity hardware using local storage as Hadoop is designed for. It was instead set up on five Blade servers (4 Hadoop nodes and a client server) using mounted NFS to the usual SB Isilon storage cluster with 14 nodes over 2 GB Ethernet. Some important properties of the cluster are show in Table 2:

---

<sup>11</sup> <http://www.myexperiment.org/workflows/3292.html>

<sup>12</sup> <http://www.myexperiment.org/workflows/4080.html>

<sup>13</sup> <http://wiki.opf-labs.org/display/SP/SB+Hadoop+Platform>

<b>Number of nodes</b>	<b>4 physical servers</b>
<b>Total number of CPU-cores</b>	48
<b>CPU specs</b>	Intel® Xeon® Processor X5670 (12M Cache, 2.93 GHz, 6.40 GT/s Intel® QPI)
<b>Total amount of RAM</b>	384 GB

Table 2 Cluster Properties

Small scale experiments have been run on the full *mp3 to wav migration and qa workflow* using the Hadoop jobs. There is an intension to run the full workflow on 1TB of input data, but for 1TB input of MP3 files, the workflow generates approximately 25TB of output and temporary WAV files. Writing this much data is a challenge for the SB Hadoop cluster set-up, where the Isilon disk I/O and local CPU use on some of the Isilon boxes were being maxed out, even though the Hadoop execution has been optimised to only run 24 maps concurrently. The 14 Isilon storage cluster nodes were first set up with two network connections. These two connections were the bottleneck. The Cluster has now been set up with five connections, which should be enough, and the full experiment will be run on this set up. Up to 14 connections can be called upon if problems still occur. It is noted that this is not a question of hardware, but of necessary connections in this rather specialized cluster set up. In the meantime a 1TB+ of "mock-up" wav input data was created for the *xcorrSound waveform-compare* Hadoop job, so that this component could be tested on its own.

### 2.5.1 Dataset

The dataset used for the *mp3 to wav migration and QA workflow* experiment is a 20 TB collection of the *Danish Radio broadcasts, mp3* experimental dataset<sup>14</sup>. The small-scale tests have been executed over a list of 58 mp3 files, totalling 7.2GB. The next stage is to execute the tests over a 1TB dataset from the collection.

The dataset used for the *xcorrSound waveform-compare Hadoop job* performance test is 1TB+ of wav input data created from the mp3 experimental dataset using a simple script. 350 pairs of wav files were generated to be compared using respectively FFmpeg and mpg321, which are the same tools used in the mp3 to wav migration workflow. The total size of this test set is 1.1TB. In this dataset we have 344 successful migrations and 6 which are "questionable".

---

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

## 2.5.2 Experiments and Results

### 2.5.2.1 *mp3 to wav migration and qa workflow small-scale experiment*

The small-scale experiments for the *mp3 to wav migration and qa workflow* were all run on a file list of **58 files (7.2Gb in total)**. Note that the launched maps are for the three Hadoop jobs FfmpegMigrate, Mpg321Convert, and WaveformCompare in order.

Max split size	Duration	Launched maps
1024	37m, 58.593s	3,3,7
256	18m, 17.917s	12,12,28
64	16m, 54.703s	47,47,113

Table 3 mp3 to wav migration results

When a map-reduce job is run, the input is divided into number of “splits”. Each split is then given as input to a map task, which will divide the split further into input key, value pairs. The max split size is the number of bytes in a split. The number of launched map tasks is the number of splits.

The exact number of launched maps seems not to have a big influence on performance, as long as there are more than 12. That is as long as max split size is at most 256 on an input file list of 58 files. It has been noted that there are approximately twice as many launched maps for the waveform-compare Hadoop job, because the input list is approximately twice as big, as it is a list of pairs. This can be adjusted to get approximately the same number of jobs, but it does not seem to affect the performance.

The first line of tests was to decide on expected optimal max split size. The next line of tests will vary on the size of the input. If max-split-size=256 (bytes) gives us  $2*12$  maps on an input-txt-file with 58 files, this means 256 bytes is approx.  $58/12=4,8333$  files, so one file is ca.  $256/4.8333=52.9655$  bytes. If approximately  $2*12$  maps are required for an input-txt-file with 1000 files, then the max-split-size would be to be approx.  $1000/12*52.9655 = 4413.7931$  bytes. This evaluation will be included in the upcoming deliverable *D18.2 SCAPE final evaluation and methodology report*.

### 2.5.2.2 *xcorrSound waveform-compare Hadoop job large-scale performance test*

To circumvent our experiment hold-up of writing too much data for our cluster set-up to handle, a 1TB+ of wav input data has been created for xcorrSound waveform-compare. 350 pairs of wav files were generated, 344 “correct” migrations and 6 “questionable”. The total size of this test set is 1.1TB.

Max split size	Duration	Success	Failure	Notes	Launched maps	NumberOfObjects PerHour
1546	58m, 18s	310	40	waveform-compare not updated to latest version	29	360
773	43m, 15s	310	40	waveform-compare not updated to latest version	57	485
580	41m, 17s	310	40	waveform-compare not updated to latest version	77	508
387	47m, 33s	310	40	waveform-compare not updated to latest version	114	441

Table 4 Results

This gives us directly  $QAFalseDifferentPercent = 40/344 = 11.6\%$ , which is not acceptable. This is because this test was run using an older version of the software. The new release 2.0.1<sup>15</sup> has fixed a bug related to white noise at the end of files, and this should get us some better correctness results. The new release has now been installed on the SB Hadoop cluster, and it is hoped to get some better correctness scores for the upcoming *D18.2 SCAPE final evaluation and methodology report*. When the mp3 to wav migration was first done the

QA workflow evaluation stated:

**Scalability** The workflow must be able to process a large collection within reasonable time. That is we want to be able to migrate and QA a large collection of radio broadcast mp3-files (20 TBytes - 175000 files) within weeks rather than years.<sup>16</sup>

While no large-scale tests for the full workflow exist, the content comparison part of the QA using xcorrSound waveform-compare can potentially be performed in weeks rather than years. If 508 files can be processed per hour, then 175000 files can potentially be processed in 15 days.

---

<sup>15</sup> <https://github.com/openplanets/scape-xcorrSound/releases/tag/2.0.1>

<sup>16</sup> <http://wiki.opf-labs.org/display/SP/EVAL-LSDR6-1>

## **2.6 Conclusion**

The main contribution of the audio QA work is the xcorrSound tool box. The workflows show that the waveform-compare tool can easily be incorporated in diverse audio QA solutions. It has been shown that a tool has been developed that compares Audio file content, and it is scalable!

## 3 Document Collections QA

### 3.1 Introduction

National libraries maintain huge archives of artefacts documenting a country's cultural, political and social history. Providing public access to valuable historic documents is often constrained by the paramount task of long-term preservation. Digitization is a mean that improves accessibility while protecting the original artefacts. Assembling and maintaining such archives of digitized book collections is a complex task. It is common that different versions of image collections with identical or near-identical content exist. Due to storage requirements it might not be possible to keep all derivate image collections. Thus the problem is to assess the quality of repeated acquisitions (e.g. digitization of documents with different equipment, settings, lighting) or different downloads of image collections that have been post processed (e.g. de-noising, compression, rescaling, cropping, etc.). A single national library produces millions of digitized pages a year. Manually cross-checking the different derivatives is an almost impossible endeavour.

Traditionally this problem was approached by using Optical Character Recognition (OCR) to compare the textual content of the digitized documents. OCR, though, is prone to many errors and highly dependent on the quality of digitized images. Different fonts, text orientations, page layouts or insufficient pre-processing complicate the process of text recognition. It also fails on handwritten and especially foreign, non-western texts. Pixel-wise comparison using digital image processing is only possible as long as no operations changing the image geometry, e.g. cropping, scaling or rotating the image were applied. Furthermore, in cases of filtering as well as colour or tone modifications the information at pixel level might differ significantly, although the image content is well preserved (see Figure 2 Matching SIFT Keypoints between two different images of the same source document). Identifying duplicates manually is a very time-consuming and error-prone process. Experts need a tool to help them: Matchbox. Matchbox is an open source tool which:

- provides decision-making support for duplicate image detection in or across collections
- identifies duplicate content, even where files are different (in format, size, rotation, cropping, colour-enhancement etc.), and if they have been scanned from different original copies of the same publication
- applies state-of-the art image processing
- works where OCR will not, for example images of handwriting or music scores
- is useful in assembling collections from multiple sources, and identifying missing files



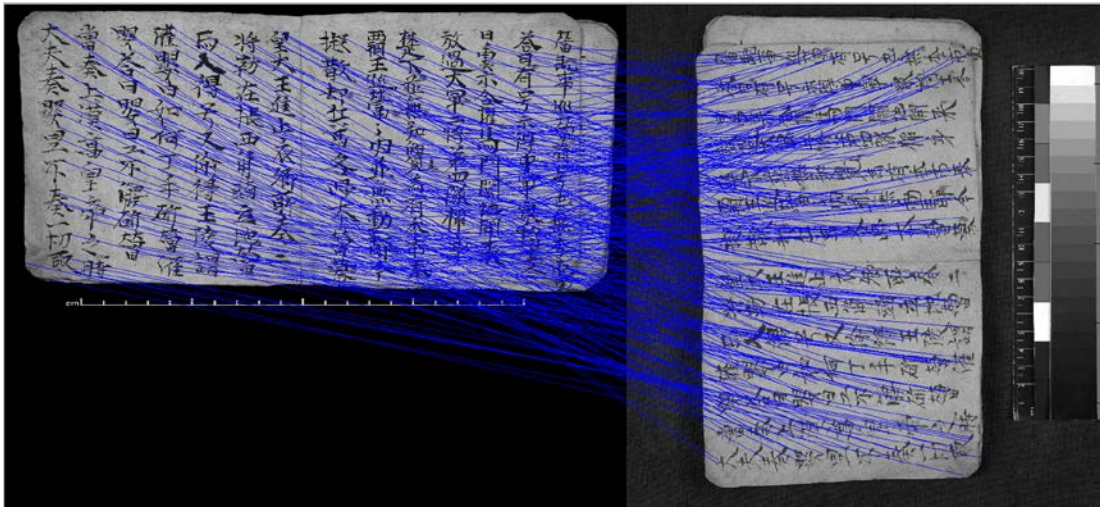


Figure 2 Matching SIFT Keypoints between two different images of the same source document

The provided Matchbox toolset can be used to efficiently detect corresponding images between different image collections as well as to assess their quality. By using inter point detection and derivation of local feature descriptors, which have proven highly invariant to geometrical, brightness and colour distortions, they are able to successfully process even problematic documents (e.g. handwritten books, ancient Chinese texts).

Scale-invariant Feature Transform (SIFT)<sup>17</sup> is an algorithm widely used in digital image processing to detect descriptive local features of images that can be used to identify corresponding points in different images. Such points are generally located at high-contrast regions. Thus, this approach is ideally for document processing because high contrast is an intrinsic property of readability. In order to be able to identify the correspondence of images between two different collections without reference to any metadata information, a comparison based on a visual dictionary, i.e. a Visual Bag of Words (BoW) approach is suggested. Following this approach a collection is represented as a set of characteristic image regions. Ignoring the spatial structure of these visual words, only their sole appearance is used to construct a visual dictionary. Using all descriptors of a collection the most common ones are selected as a representative bag of visual words. Each image in both collections is then characterized by a very compact representation, i.e. a visual histogram counting the frequencies of visual words in the image. By means of such histograms different collections can be evaluated for correspondence, duplication or absence of images. A further option is the detection of duplicates within the same collection, with the exception that a visual dictionary specific to this collection is constructed beforehand. The Visual BoW is related to the BoW model in natural language processing, where the bag is constructed from real words.

<sup>17</sup> [http://en.wikipedia.org/wiki/Scale-invariant\\_feature\\_transform](http://en.wikipedia.org/wiki/Scale-invariant_feature_transform)

Having solved the image correspondence problem using the visual BoW, a detailed comparison of image pairs, i.e. an all-pairs problem with respect to all pairs of possibly matching descriptors between two images, is performed. Corresponding images are processed using descriptor matching, affine transformation estimation, image warping and computation of the structural similarity between the images.

## 3.2 Tools and Interfaces

The Matchbox Toolset<sup>18</sup> is a collection of small command line tools that are intended to work independently. Matchbox brings the following benefits:

- Automated quality assurance
- Reduced manual effort and error
- Saved time
- Lower costs, e.g. storage, effort
- Open source, standalone tool. Also as Taverna component for easy invocation
- Invariant to format, rotation, scale, translation, illumination, resolution, cropping, warping and distortions
- May be applied to wide range of image collections, not just print images

These tools comply with the standard pattern of a typical retrieval task. At first descriptive information has to be extracted from unstructured data. In this case this corresponds to extracting information from images. The extracted features are typically numbers in vector format and are the basis for further calculations. The previously unstructured and huge amounts of data have been transformed into a condensed vector space representation upon which various machine learning techniques can be applied for further processing. Depending on the scenario these techniques can be used to classify the images into different categories, find duplicates or compare different collections.

The Matchbox Toolset consists of three executables: `extractfeatures`, `train` and `compare`. Each of them is responsible for a certain task within a workflow that is described by a scenario.

### **extractfeatures**

The command line tool `extractfeatures` can be used to extract different image processing features and store them in a machine readable format. By default, extracted data is stored in zipped xml format to maintain cross-compatibility with other tools.

The following features are supported by Matchbox:

---

<sup>18</sup> <http://openplanets.github.io/matchbox/>

Image Metadata: A small set of image metadata is extracted. The emphasis on this feature set has been dropped because this work package provides further tools that specialize on metadata.

Image Histogram: These features are simple colour histograms that are traditionally used in image processing to describe the distribution of colour within the image.

Image Profile: Image profiles are vertical and horizontal projections and represent the average colour intensity value for each pixel row or column.

SIFT: Scale Invariant Feature Transforms describe local features in images that are invariant to changes in scale, rotation, illumination and viewpoint.

BoW Histogram: Bag of Words Histograms are subsequently created from mapping all SIFT key points of an image to a previously calculated visual vocabulary.

### **train**

This command line tool is used to create the visual vocabulary. It takes SIFT features as input that have been previously extracted from each image of a collection using the *extractfeatures* tool. *train* loads these features and creates a visual Bag of Words (BoW). By applying a k-means clustering algorithm<sup>19</sup> that identifies a given number of cluster centers within a multidimensional vector space, the tool creates the descriptive vocabulary from these centers and stores it to a XML-file.

### **compare**

This command line tool can be used to compare images by their extracted features. Depending on the type of features used for comparison, either distance metrics or similarity measures are calculated.

The results are outputted to standard output which is typically the screen. This output can be relayed into another tool (e.g. decision making). Duplicate detection is the task of detecting duplicates within an image collection.

- extract SIFTComparison features of all images
- train a visual vocabulary on the extracted features
- extract BoWHistograms using the vocabulary and all extracted SIFTComparison features
- create a similarity matrix for the collection using compare on all BoWHistogram features
- take the top-most similar images for each image and compare their SIFTComparison features
- Set a threshold based on the retrieved data
- images with an SSIM exceeding the threshold are considered to be duplicates

---

<sup>19</sup> [http://en.wikipedia.org/wiki/K-means\\_clustering](http://en.wikipedia.org/wiki/K-means_clustering)

To build the Matchbox Toolset from source, download the sources from GitHub:

<https://github.com/openplanets/matchbox>

Further instructions on how to compile Matchbox from source are provided in the INSTALL file in the source tree.

### 3.2.1 Scalability

The scalability was realized by means of Hadoop platform and Python scripts  
<https://github.com/openplanets/matchbox/tree/master/hadoop/pythonwf>.

- The working directory where scripts are located e.g. pythonwf should be defined by an expert.
- The HDFS file system should contain directory with image collection (default name is "collection") that should be analysed.
- The path to the working collection can be passed as a parameter to the script.
- In order to leverage Matchbox algorithms the image collection should have at least 10 files related to bow size (explained below).
- The default bow size of the bag of visual words (dictionary) is a 1000 (defined in CalculateBow.sh). Having small collection (< then 10 images) it will not be possible to calculate with this bow size. The bow size (e.g. 100) can be reduced for small collection but it will reduce the quality of analysis.
- Outputs are expected in "matchbox" directory on HDFS (/user/training/matchbox/summary.csv e.g. 00000032;00000034;0.748908 means that image 00000032.jp2 is similar to the image 00000034 with similarity score 0.748908. Similarity score is between 1 and 0, where 1 means best similarity and 0 means no similarity)
- Each workflow step is a precondition for the next step

Note that the dfs permissions should be disabled by extension in conf/hdfs-site.xml with

```
<property>  
  
<name>dfs.permissions</name>  
  
<value>false</value>
```

The workflow (see Figure 4) is implemented using Python scripts. In order to start full Matchbox Python Hadoop workflow use:

```
./PythonMatchboxWF.sh
```

Optional parameters are:

1. local home path e.g. `"/home/training/pythonwf/"`
2. HDFS home path e.g. `"/user/training/"`
3. HDFS input directory where image collection is stored e.g. `"collection"`

Example with parameters:

```
./PythonMatchboxWF.sh /home/training/pythonwf /user/training collection
```

or the following steps can be used:

Create manifest file with paths to the input files. Input dir parameter possible:

```
./CreateInputFiles.sh
```

Execute extract features step:

```
./CmdExtractSift.sh
```

Calculate BoW step:

```
./CmdCalculateBoW.sh
```

Extract visual histogram step:

```
./CmdExtractHistogram.s
```

Compare visual histograms, find three nearest neighbours and perform SSIM:

```
./CmdCompare.sh
```

### **3.3 Taverna Workflows**

The Taverna workflow engine wraps the Matchbox tool installed on remote Linux VM server for duplicate search in digital document collection using Hadoop streaming API. The result of duplicate

search is returned in text format. This tool is a part of DPQAlib library <sup>20</sup>, is written in C++ and provided as a Python script using associated ZIP on Windows or Debian package on Linux and can be found:<https://github.com/openplanets/matchbox/master/tree/DPQAlib>

The workflow MatchboxHadoopApi.t2flow<sup>21</sup> enables using of matchbox tool on Hadoop with Taverna. This workflow is based on Python scripts and Hadoop Streaming API included in "pythonwf" folder of pc-qa-matchbox project on GitHub<sup>22</sup>. Mentioned Python scripts can also be used without Taverna. Taverna is just a wrapper for these scripts for convenience.

For this workflow we assume that digital collection is located on HDFS and we have a list of input files in format "hdfs:///user/training/collection/00000032.jp2&quot; - one row per file entry.

This list can be also generated in scripts. Changing parameters user can customize the workflow and adjust it to the institutional needs.

This workflow does not apply ToMaR and uses directly Hadoop Streaming API to avoid additional dependencies. The workflow has four input parameters but could be also used with default parameters. These parameters are:

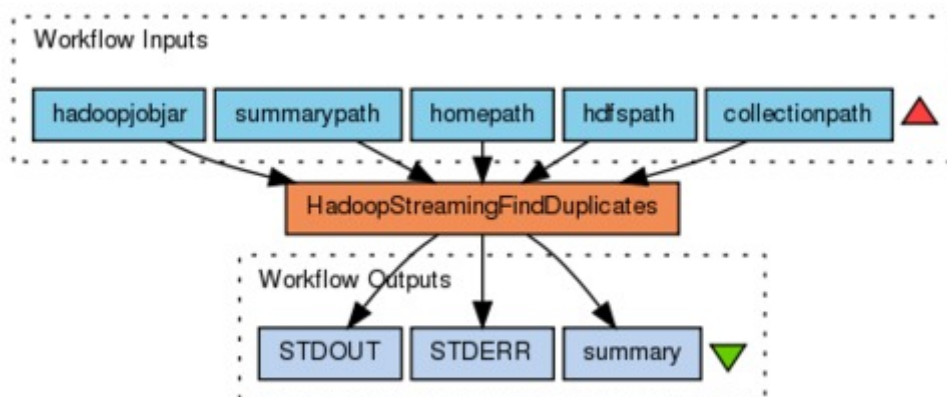


Figure 3 Taverna Workflow for Matchbox tool using Hadoop scalability workflow

1. homepath is a path to the scripts on a local machine e.g. "/home/training/pythonwf"
2. hdfspath is a path to the home directory on HDFS e.g. "/user/training"
3. collectionpath is a name of the folder that comprises digital collection on HDFS e.g. "collection"

<sup>20</sup> <https://github.com/openplanets/scape/tree/master/pc-qa-matchbox/DPQAlib>

<sup>21</sup> <http://www.myexperiment.org/workflows/3892.html>

<sup>22</sup> <https://github.com/openplanets/scape/tree/master/pc-qa-matchbox/hadoop/pythonwf>

4. summarypath is a name of the folder that comprises calculation results (list of possible duplicates) on HDFS e.g. "compare".

The list of possible duplicates can be found in file benchmark\_result\_list.csv in summary path. The main script in a workflow is a PythonMatchboxWF.sh that comprises all other scripts. An experienced user could execute each workflow step in a separate module in order to better manage script parameters.

1. The first step in the workflow is a preparation of input files list and is performed by CreateInputFiles.sh. Result of this step is a file with paths to collection files stored on HDFS in input files folder.
2. The second step is a SIFT features extraction calculated using "binary" parameter in order to improve performance. Result of this step are feature files for each input file like
3. "00000031.jp2.SIFTComparison.descriptors.dat",  
"00000031.jp2.SIFTComparison.keypoints.dat" and  
"00000031.jp2.SIFTComparison.feats.xml.gz" stored in matchbox folder on HDFS.
4. The third step is a calculation of Bag of Words (visual dictionary) performed by CmdCalculateBoW.sh. Result is stored in bow folder in bow.xml file on HDFS.
5. Then the visual histograms are extracted using CmdExtractHistogram.sh for each input file. Result is stored in folder histogram on HDFS e.g. "00000031.jp2.BOWHistogram.feats.xml.gz".
6. The final step is to perform actual comparison using CmdCompare.sh. Results are stored in compare folder on HDFS and comprise file benchmarkk\_result\_list.csv that presents possible duplicates - one pair per row e.g. img1;img2;similarity between 0 (low) and 1 (high)

### 3.4 Demos

Matchbox has also online demo page<sup>23</sup> that can be used to test the tool. It can be accessed by following the link : <http://scape.opf-labs.org/matchbox/>

### 3.5 Benchmarking

There are numerous situations in which you may need to identify duplicate images in collections, for example:

- to ensure that a page or book has not been digitized twice
- to discover whether a master and service set of digitized images represent the same set of originals
- to confirm that all scans have gone through post-scan image processing

---

<sup>23</sup> <https://github.com/openplanets/scape-demo-sites/tree/master/matchbox>

Correctness of the Matchbox toolset will be evaluated on scenario  *LSDRT11 Duplicate image detection within one book* <sup>24</sup>. The corresponding workflow has been implemented as an executable script in the programming language Python. The workflow is described in Figure 4. The dataset described in Section 3.5.1. The implemented workflow will process every collection using the Matchbox toolset and searches for duplicates. The result will be evaluated against the ground truth data. Accuracy will be based on how many duplicates are detected correctly.

1. Document feature extraction
  - Interest key points - Scale Invariant Feature Transform (SIFT)
  - Local feature descriptors (invariant to geometrical distortions)
2. Learning visual dictionary
  - Clustering method applied to all SIFT descriptors of all images using k-means algorithm
  - Collect local descriptors in a visual dictionary using Bag-Of-Words (BoW) algorithm
3. Create visual histogram for each image document
4. Detect similar images based on visual histogram and local descriptors. Structural SIMilarity (SSIM) approach
  - Rotate
  - Scale
  - Mask
  - Overlaying

---

<sup>24</sup> <http://wiki.opf-labs.org/display/SP/LSDRT11+Duplicate+image+detection+within+one+book>



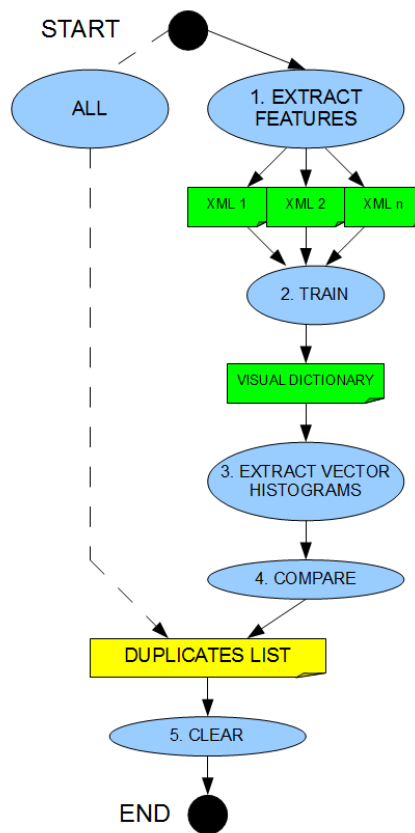


Figure 4: Duplicate detection workflow

### 3.5.1 Dataset

The benchmark dataset consists of 59 digitized books provided by the Austrian National Library. It is a presentable example of content that has to be processed in the different scenarios. Additionally it is a good example against the use of OCR algorithms. The dataset contains examples for the following content classes:

- normal justified article pages
- multi-column pages
- handwritten content
- Tables
- mixed content
  - printed and handwritten letters
  - printed pages with handwritten annotations
  - Images
- Different languages

- Books in Cyrillic, Greek, Hebrew language
- Mixed languages (side by side, paragraphs, textboxes)

Examples from this dataset:



### 3.5.2 Experiments and Results

Table 5 describes the environment for the experiments and shows performance results.

Code for Hadoop on GitHub	<a href="https://github.com/openplanets/matchbox/tree/master/hadoop/pythonwf">https://github.com/openplanets/matchbox/tree/master/hadoop/pythonwf</a>
Description of test dataset	Selected files with ground truth, 10 pages / JP2 images
Link to test Dataset	<a href="http://wiki.opf-labs.org/display/SP/Austrian+National+Library+-+Digital+Book+Collection">http://wiki.opf-labs.org/display/SP/Austrian+National+Library+-+Digital+Book+Collection</a>
Number of nodes	1 master
Total number of CPU-cores	2
CPU specs	Intel Core i7-3520M 2.90GHz 64-bit
Total amount of RAM in GBytes	8
NumberOfObjectsPerHour	73
AverageRuntimePerItemInHours (sec)	49,315
Average Document Size (KB)	402,44

Table 5 Performance tests settings and results

### **3.6 Conclusion**

The Matchbox tool provides the decision making support for duplicate detection in document image collections. An automatic approach delivers a significant improvement when compared to manual analysis. The tool is available as Python scripts and Taverna components for easy invocation and testing. System ensures quality of the digitized content and supports managers of libraries and archives with regard to long-term digital preservation.

## 4 Web QA

### 4.1 Introduction

Archiving and preserving the Web is becoming more and more crucial for preservation bodies such as archives and libraries. But web archiving still goes together with a number of challenges. It is usually performed using Web crawlers (bots) that capture Web resources by parsing Web pages and store these into an archiving format called (W)ARC<sup>25</sup>. An access tool, such as the Internet Archive Wayback machine, is then plugged to the storage infrastructure and allows navigation within captured content with the live look and feel. These processes work fine when archiving simple websites but many technical challenges remain when trying to capture more complex/dynamic ones.

Web archiving teams therefore face the constant need to optimize crawling and access tools to overcome new technological challenges and manage the ephemerality of content published on the web. As the web is growing in size as well as in complexity, institutions in charge of web archiving are also forced to review their crawling strategies to tackle scalability issues. Developing scalable tools and methods at all stages of web archiving projects is therefore crucial. Among these steps, the quality control of web archives is currently one of the most manual and is therefore very costly and time consuming. To solve this issue, the Web QA task looked into providing a tool that would replicate the human action of checking the quality of web content crawled by looking at a sample of archived web pages and by comparing these to their live version.

The Pagelyzer tool described below, allows an automated comparison of two web pages and provides a similarity score as a human assessor would do.

### 4.2 Tools and Interfaces

Pagelyzer is a supervised framework developed to compare two web page versions based on their visual renderings, their content or both of them (hybrid). N.B. The aim is not to compare two web pages totally different (however, Pagelyzer also can do that) but web page versions. It is based on:

- a combination of structural and visual comparison methods embedded in a statistical discriminative model,
- a visual similarity measure designed for Web pages that improves change detection,
- a supervised feature selection method adapted to Web archiving.

---

<sup>25</sup> <http://www.digitalpreservation.gov/formats/fdd/fdd000236.shtml>

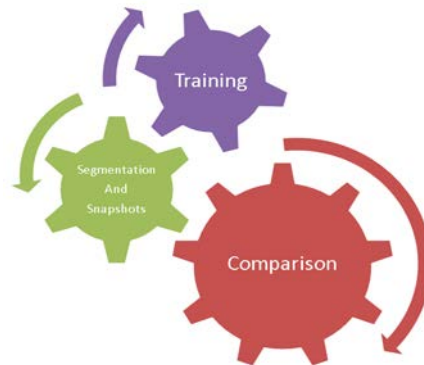


Figure 5 Pagelyzer tools

#### 4.2.1 Pagelyzer

Pagelyzer consists of three tools as shown in Figure 5: Training, Segmentation and Snapshots and Comparison. Details for each tool:

##### 4.2.1.1 Training

Pagelyzer is a supervised framework which means that by training, the system is told what is similar and what is not similar according to your needs by using an annotated dataset. This dataset should contain pairs of URLs annotated by humans based on the dataset training follows the steps:

1. It calculates the similarity (or distance) as a vector based on the comparison type. If it is image-based, your vector will contain the features related to images similarities (e.g. SIFT<sup>26</sup>). If it is content-based, your vector will contain features for text similarities (e.g. cosine similarity for links, images and words). It puts all of these vectors into vector space as seen in Figure 6. (It is an example of a two-dimensional vector). Red ones are the URL pairs annotated as dissimilar; the green ones are annotated as similar.

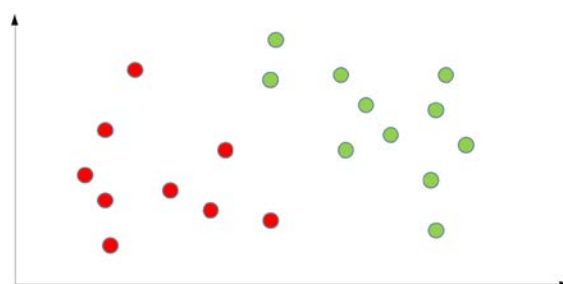


Figure 6 Annotated data on the vector space

---

<sup>26</sup> [http://en.wikipedia.org/wiki/Scale-invariant\\_feature\\_transform](http://en.wikipedia.org/wiki/Scale-invariant_feature_transform)

2. The optimal decision boundary (hyper plane) is found in input space. Anything above the decision boundary should have label 1 (similar). Similarly, anything below the decision boundary should have label 0 (dissimilar)
3. The information related to the decision boundary is saved into an output file in order to be used in the comparison step.

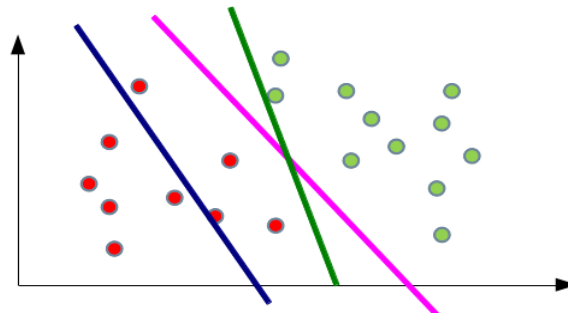


Figure 7: Finding decision boundary

#### 4.2.1.2 Segmentation and Snapshots

For training and also for comparison, the features need to be defined in order to create vectors. For the image based comparison, the screenshot of the two URLs are taken and 4 SIFT features are extracted to create vectors. For content based comparison, instead of comparing whole web pages, the web pages are segmented into blocks which are compared instead of whole page. 6 features are extracted (one for hyperlink source, one for hyperlink anchor, one for image object source, one for image anchor, one for text) to create a vector. For hybrid, the vector contains both features of content and image.

To take a snapshot of the pages and to do the segmentation, a tool called BOM (Block-o-Matic)<sup>27</sup> was developed. This tool takes the snapshot of the pages by using Selenium<sup>28</sup> and also does the segmentation of the web pages by using Javascript injection. Presented here is the detail for the Block-o-Matic web page segmentation algorithm used by Pagelyzer to perform the segmentation. It is a hybrid between the visual-based approach and document processing approach.

The segmentation process is divided in three phases: analysis, understanding and reconstruction. It comprises three tasks: filter, mapping and combine. It produces three structures: DOM structure, content structure and logic structure. The main aspect of the whole process is producing the structure where the logic structure represents the final segmentation of the web page. The DOM tree is obtained from the rendering of a web browser. The result of the analysis phase is the content

<sup>27</sup> <http://www-poleia.lip6.fr/~sanojaa/BOM/>

<sup>28</sup> <http://docs.seleniumhq.org/>

structure (Wcont), built from the DOM tree with the d2c algorithm<sup>29</sup>. Mapping the content structure into a logical structure (Wlog) is called document understanding. This mapping is performed by the c2l algorithm<sup>29</sup> with a granularity parameter pG. Web page reconstruction gathers the three structures (Rec function),

$$W' = \text{Rec}(\text{DOM}, \text{d2c}(\text{DOM}), \text{c2l}(\text{d2c}(\text{DOM}, \text{pG}))).$$

For the integration of the segmentation outcome to Pagelyzer an XML representation is used: ViXML. It represents hierarchically the blocks, their geometric properties, the links and text in each block.

#### 4.2.1.3 Comparison

The system is ready to compare/classify new URLs. When you have new pair of URLs without any annotation, based on the decision boundary<sup>30</sup>, it can be determined if they are similar or not. By using the vectors created by using different features, the pairs of URLs are places in the vector space and the system can decide if they are similar or not based on their place related to the decision boundary. In Figure 8 the pair of URLs in blue will be considered as dissimilar, the ones in black will be considered as similar by Pagelyzer.

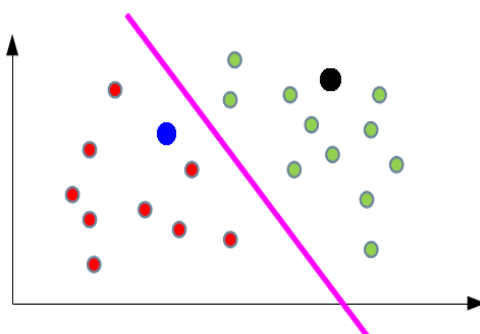


Figure 8 Example of decision

#### 4.2.2 Scalability

As will be discussed further in Section 4.5, the comparison of two web pages is very time consuming – seconds per comparison. Therefore, considering the amount of pairs of candidate documents to

<sup>29</sup> A.Sanoja, S.Gnaçarski Block-o-Matic : a Web Page Segmentation Tool and its Evaluation, 2013, <http://www.hal.archives-ouvertes.fr/hal-00881693>

<sup>30</sup> In a statistical-classification problem with two classes, a decision boundary or decision surface is a hypersurface that partitions the underlying vector space into two sets, one for each class. The classifier will classify all the points on one side of the decision boundary as belonging to one class and all those on the other side as belonging to the other class. [http://en.wikipedia.org/wiki/Decision\\_boundary](http://en.wikipedia.org/wiki/Decision_boundary)

compare, a mechanism to provide a way to speed the comparison up is required. Since the comparison algorithm itself is not parallelisable, the focus was on providing a tool to run several comparisons in parallel.

At Internet Memory, distribution wrapper has been developed that leverages the heart of SCAPE architecture – MapReduce – to distribute this time intensive task. In the deployment scenario the browsershot Selenium framework was decoupled from the segmentation computation and the comparison score calculation. The comparison calculations are run directly on the MapReduce worker nodes, Selenium is run on a separate set of nodes and the JS segmentation computation scripts are hosted again on a separate node.

The scalability factor anticipated is linear, provided that the biggest bottleneck, the Selenium network, will keep up with the number of nodes asking for the browser shots. Two dedicated machines (one for Firefox instance and one for Opera browser) and two worker nodes each having 3 map slots were used. The input to the experiment is a set of URLs – described in detail in Section 4.5.1, where each is rendered using two browser types (Firefox and Opera) and then their images and segmentations are compared using Pagelyzer tool. The scores are recorded and exported via C3P0<sup>31</sup> adaptor to SCOUT<sup>32</sup>. The results provided by the experiment evaluation confirmed the anticipation of the linear scaling. The parallelised browsershot tool, can be checked out from the public Github repository at <https://github.com/openplanets/browser-shot-tool-mapred>.

### 4.3 Taverna Workflows

The full Taverna Workflow is available from myExperiment and generated by using tool-wrapper: <http://www.myexperiment.org/workflows/4238.html>

In detail, the workflow takes a configuration file path, and two URLs: url1 and url2 and returns a score between -1 and 1 to decide if two pages (url1 and url2) are similar or not.

---

<sup>31</sup> <http://ifs.tuwien.ac.at/imp/c3po>

<sup>32</sup> <http://scout.scape.keep.pt/web/>



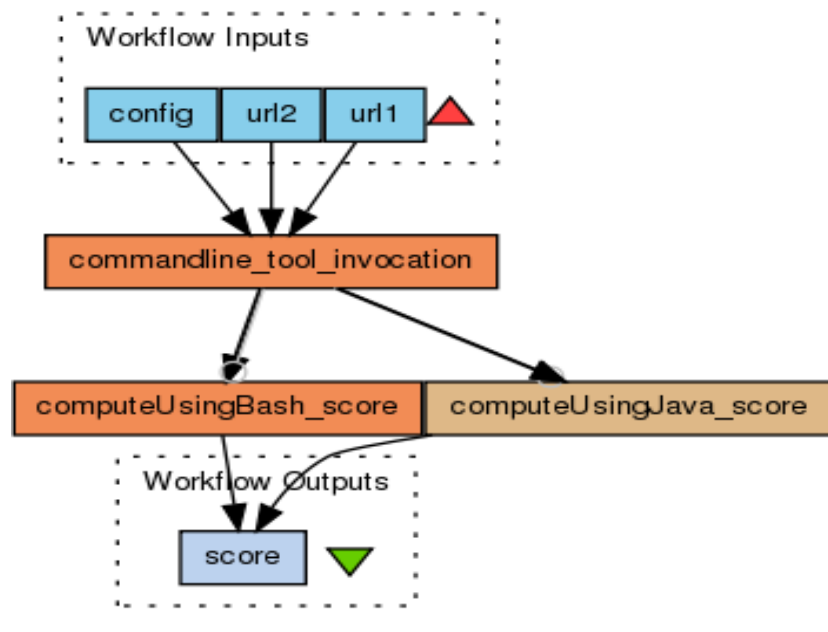


Figure 9 Pagelyzer Taverna workflow

## 4.4 Demos

Pagelyzer has also online demo page<sup>33</sup> that can be used to test the tool. It can be accessed by following two links:

<http://scape.lip6.fr/pagelyzer/>  
<http://scape.opf-labs.org/pagelyzer/>

## 4.5 Benchmarking

This section contains the performance based benchmarking results. Correctness based benchmarking details can be found in MS54 report<sup>34</sup>.

### 4.5.1 Dataset

The dataset used for our test is composed of URLs of web pages archived within the Internet Memory Foundation web archive and freely accessible online. As for tests conducted within deliverable D11.2, we selected randomly URLs from the web archive and compared their rendering

<sup>33</sup> <https://github.com/openplanets/scape-demo-sites/tree/master/pagelyzer>

<sup>34</sup> [https://portal.ait.ac.at/sites/Scape/Management/Lists/Milestones List/DispForm.aspx?ID=54](https://portal.ait.ac.at/sites/Scape/Management/Lists/Milestones%20List/DispForm.aspx?ID=54)

through two access tools allowing rendering of archived web pages as on the live website. The first access tool is the one currently in use within our production environment and the other one is the new version about to be added within our production workflow. Comparing two versions of the archived web pages rather than comparing the live and the archive version provides more stability to our tests, in the sense that versions remain in the same status and allow to extend the test period if required. For each URL annotated, assessors also provided a description of the type of content viewed (html, js, CSS, etc.) as it was pointed by University Pierre and Marie Curie(UPMC) as having an impact on the comparison tool accuracy. Table 6 below shows the repartition of content type within the corpus used, as annotated by the assessors.

Content Type	Number	Percentage
HTML	273	60.80
Index page	52	12.69
Blank page	57	11.58
Js file	2	6.46
Error message	29	3.34
Error page	15	2.67
Xml file	6	1.34
Image	1	0.45
Zip file	1	0.22
Doc file	1	0.22
PDF file	1	0.22

Table 6 Content type repartition

**4.5.2 Experiments and Results**

Pagelyzer performance depends on the comparison mode. For image based comparison, simply taking the snapshot is necessary; for content based comparison, only the segmentation step is used; for hybrid (content and image based) both steps, screenshot and segmentation, are necessary. Thus, we can say that the hybrid comparison gives us the worst case comparison performance. The performance changes also based on the web page structure, more content, more time for

segmentation. That's why the results presented here for one pair of URLs is an average over the whole dataset.

Capture One URL			Getting Score	Overall
3500ms			819ms	8075ms
Segmentation	Screenshot	Selenium init		
0.13 %	0.17%	0.71%		

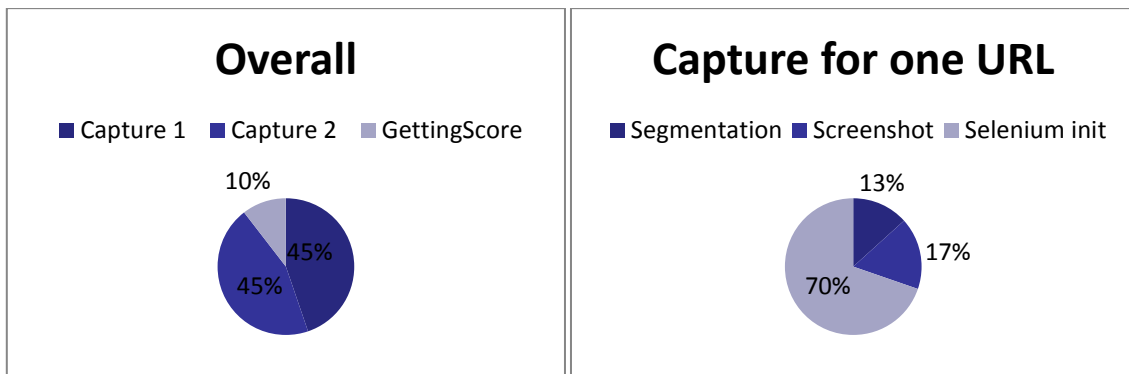


Table 7 Benchmarking Results for Pagelyzer Hybrid Mode

Description of test dataset	499 pairs of URL provided by IMF
Number of nodes	1
Total number of CPU-cores	1
CPU specs	Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz
Total amount of RAM in GBytes	16
NumberOfObjectsPerHour	3600/8=450
AverageRuntimePerItemInHours	0.0022

Table 8 Description of dataset and execution environment

## **4.6 Conclusion**

As outlined within the introduction, quality assurance applied to web archives is currently difficult to perform. This is even more complex when crawling and hosting very large amount of web data. Indeed, if methods and tools exists, such as automated check of 404's or crawl statistics review, these are most of the time very costly in time and human resources. Furthermore, these methods do not allow checking of the rendering of web pages archived in an automated manner and this is therefore most of the time made manually when done at all, on small samples.

It is emphasized that Pagelyzer is a research project and different approaches are still being developed such as giving an importance value to each of block based on their place on the web pages etc. that could be of great interest for web archives.

## 5 Image QA

This section describes different QA tools for different types of images.

### 5.1 Jpylyzer

Jpylyzer is a validator and feature extractor for JP2 (JPEG 2000 Part 1 / ISO/IEC 15444-1) images. This tool serves two purposes:

1. Format validation – does an image really conform to the format's specifications?
2. Feature extraction - what are an image's technical characteristics?

Within SCAPE, answering the above questions is particularly important in the case of large-scale TIFF to JP2 image migration workflows<sup>35</sup>. At the start of the project it became clear that JHOVE<sup>36</sup>, which is the only existing tool with similar functionality, had shortcomings that severely limit its usefulness in imaging workflows<sup>37</sup>. Also, JHOVE's development at the time had come to a standstill due to its replacement by JHOVE2<sup>38</sup> (which doesn't include a JPEG 2000 module at all), and as a result its future was rather unclear. An early attempt at a rudimentary JP2 file structure checker<sup>39</sup> indicated that developing a full-fledged validator and feature extractor would be entirely feasible, and this ultimately resulted in the development of the Jpylyzer tool.

#### 5.1.1 Functionality and scope

Within the context of imaging workflows, Jpylyzer's validation functionality can be used to ensure that created images are standards-compliant; besides, it will also detect common types of byte-level corruption. Examples are truncated or otherwise incomplete images, or images that are affected by file transfer errors (e.g. images that were mistakenly transferred over a network connection in ASCII mode). The feature extraction functionality makes it possible to test whether a JP2 conforms to a technical profile. For example, Jpylyzer's output gives detailed information on whether an image uses lossless or lossy compression, its compression ratio, colour space, resolution, and so on.

---

<sup>35</sup> See: <http://wiki.opf-labs.org/display/SP/Large+Scale+Image+Migration>

<sup>36</sup> Link: <http://jhove.sourceforge.net/>

<sup>37</sup> Details on JHOVE's shortcomings are provided here: <http://www.openplanetsfoundation.org/blogs/2011-09-01-simple-jp2-file-structure-checker>

<sup>38</sup> Link: <https://bitbucket.org/jhove2/main/wiki/Home>

<sup>39</sup> Described in: <http://www.openplanetsfoundation.org/blogs/2011-09-01-simple-jp2-file-structure-checker>

One important limitation is that Jpylyzer does not analyse the data in the compressed bitstream segments. This would involve decoding the whole image, which is completely out of Jpylyzer's scope. This means that it is possible that a JP2 that passes each of Jpylyzer's tests will nevertheless fail to render correctly in a viewer application. For the latter, it is recommended that the image QA workflow also includes some pixel-based image comparison between the source and destination images (in case of a format migration), or otherwise at least a test that checks if an image renders at all. Depending on the specific needs, this can be done with software such as ImageMagick or the SCAPE tool Matchbox.

### **5.1.2 Use and deployment**

Jpylyzer has its own microsite that provides links to source code, binaries for Linux- and Windows-based systems. It is available from the following link:

<http://openplanets.github.io/jpylyzer/>

The microsite also includes an exhaustive User Manual, which is available here:

<http://openplanets.github.io/jpylyzer/userManual.html>

The User Manual covers all aspects related to installing and using Jpylyzer, including a detailed description of every single test that Jpylyzer performs for validation, as well as every reported property.

### **5.1.3 Workflows**

#### ***5.1.3.1 Detection of damaged JP2s at British Library***

JISC 1 was a project funded by the Joint Information System Committee, which digitized 275,000 19th Century newspaper issues (2 million images) that are held by The British Library (BL). The image masters were initially stored as uncompressed TIFF, but in order to save money on storage costs these were later converted to the JP2 format. This reduced the size of the master image collection from 80 to 45 TB. The conversion was implemented as an ingest workflow that used JHOVE 1.6 to validate the converted image before ingest into the repository. Sometime after ingest, a number of images turned out to be malformed, even though JHOVE had indicated that these files were valid.

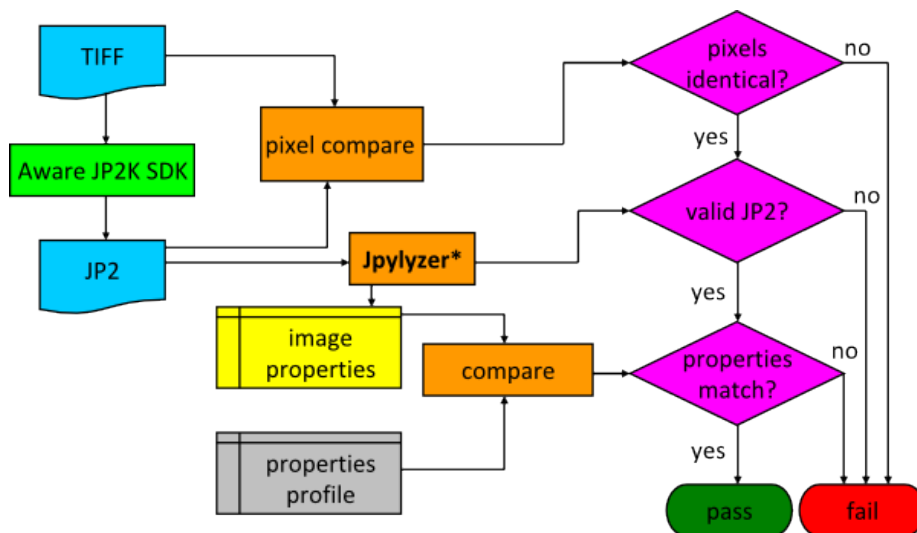
A test on a sample set demonstrated that Jpylyzer was capable of identifying every broken image. Jpylyzer was then used to validate every single master image in the collection, which yielded 676 invalid JP2s in the entire collection. Since the TIFF originals were still available, these were then used to re-generate the broken JP2s. Jpylyzer was also used to test two more collections (JISC2 and 19th Century books). This revealed another 3 invalid JP2s for JISC2, and none for the 19th Century books.

### 5.1.3.2 Metamorfoze TIFF to JP2 migration

'Metamorfoze' is the Netherlands' national program for the preservation of paper heritage. It is a collaborative effort of the National Library of the Netherlands (KB) and the National Archives of the Netherlands. It employs digitization as one of its conservation methods (preservation imaging). Until recently, Metamorfoze has been using uncompressed TIFF as its preservation format. This has resulted in about 1460 TB worth of TIFF images being stored at the KB by the end of 2012. In order to reduce storage costs, the KB decided to migrate these images to lossless JP2. One particular risk of such a large scale migration is that hardware failure may result in corrupted images. Since some of the Metamorfoze material is irreplaceable (because the paper originals are in poor shape), it is vital to have a workflow that includes checks that ensure the integrity of the migrated images.

To this end, a workflow was set up that uses Jpylyzer to verify that each created image is valid and intact JP2. In addition, Jpylyzer's feature extraction output is used to check that each image's encoding options (compression type, number of decomposition levels and quality layers, progression order, tile- and codeblock size, error resilience markers) match a pre-defined profile. Finally, a pixel-wise comparison was done between each JP2 and its source TIFF image. Figure 10 gives an overview of the imaging workflow.

This work was not part of SCAPE; however, a small (8047 images) subset of the Metamorfoze collection was used for an experiment with a pseudo-distributed Hadoop migration workflow. This is described in further detail in Section 5.1.3.3.



*\*Imported as module in Python-based workflow*

Figure 10: Imaging workflow for Metamorfoze TIFF to JP2 migration

### **5.1.3.3 Validation of digitized newspapers**

The Danish State and University Library (SB) ran an experiment that used Jpylyzer for validating 17978 JP2s (167 GB) of digitized newspaper pages<sup>40</sup>. The experiment involved validating each newspaper page against the JP2 specification, as well as a further assessment of the extracted properties against an institutional profile, which is linked to a policy. A detailed description of the experiment can be found here:

<http://wiki.opf-labs.org/display/SP/Validate+JPEG2000+Newspapers+Using+jpylyzer>

The validation was implemented as a Hadoop workflow on the SB's Hadoop Platform<sup>41</sup>. Before running the experiment, the aim was to be able to process at least 5000 pages per hour. At 65000 pages per hour, the actual performance exceeded this target by a factor 13. With this in mind, nothing speaks against using Jpylyzer in large, scalable workflows. A detailed evaluation of this experiment is available here: <http://wiki.opf-labs.org/display/SP/Evaluation+1+-+JPEG2000+validation>

### **5.1.3.4 Quality control at the Wellcome Library**

The Wellcome Library uses Jpylyzer for validating JP2 images, both those that are produced internally as well as those received from external suppliers<sup>42</sup>. Their imaging workflow is based on the Goobi software (developed by Intranada GmbH), which uses Jpylyzer as a plug-in<sup>43</sup>.

### **5.1.3.5 Taverna workflows**

Within SCAPE Jpylyzer is used in a number of Taverna workflows. Notable examples are the two TIFF to JP2 workflows by the Austrian National Library, one of them also includes an image comparison step that uses Matchbox. The British Library created also workflow that uses ImageMagick for the image comparison<sup>44</sup>. Figure 11 gives a graphical representation of the above workflow.

---

<sup>40</sup> <http://wiki.opf-labs.org/display/SP/Danish+newspaper+-+Morgenavisen+Jyllandsposten>

<sup>41</sup> Described in detail here: <http://wiki.opf-labs.org/display/SP/SB+Hadoop+Platform>

<sup>42</sup> [http://www.slideshare.net/goobi\\_org/goobi-in-the-wellcome-library](http://www.slideshare.net/goobi_org/goobi-in-the-wellcome-library)

<sup>43</sup> <http://www.digiverso.com/en/products/goobi/history/42-products/goobi/329-history-goobi-1-9-2-intranada-edition>

<sup>44</sup> <http://www.myexperiment.org/workflows/3401.html> ,<http://www.myexperiment.org/workflows/4276.html>



#### 5.1.4 Demos

A video that gives a brief overview of Jpylyzer has been prepared<sup>45</sup>. The link below is a simple online demo that allows you to upload a JP2 and have it validated by Jpylyzer. Note that the properties that are shown in this demo only represent a small subset of Jpylyzer's full output.

<http://scape.opf-labs.org/jpylyzer/>

#### 5.1.5 Benchmarking

##### 5.1.5.1 *Performance based benchmarks*

As part of Work Package TB.WP2 (Large Scale Digital Repositories Testbed), an experimental TIFF to JP2 migration workflow was set up and implemented as a pseudo-distributed Hadoop workflow<sup>46</sup>. This was adapted from the KB's Metamorfoze migration workflow (which was described in section 5.1.3.2), and uses a small subset (8047 images) of the Metamorfoze collection. The results of this experiment serve as a useful illustration of Jpylyzer's performance. Table 7 lists the general characteristics of the dataset and the environment that was used to run the experiment.

---

<sup>45</sup> <http://vimeo.com/53693082>

<sup>46</sup> The workflow is *pseudo*-distributed because the total number of physical CPUs on the KB Hadoop platform is only 1!

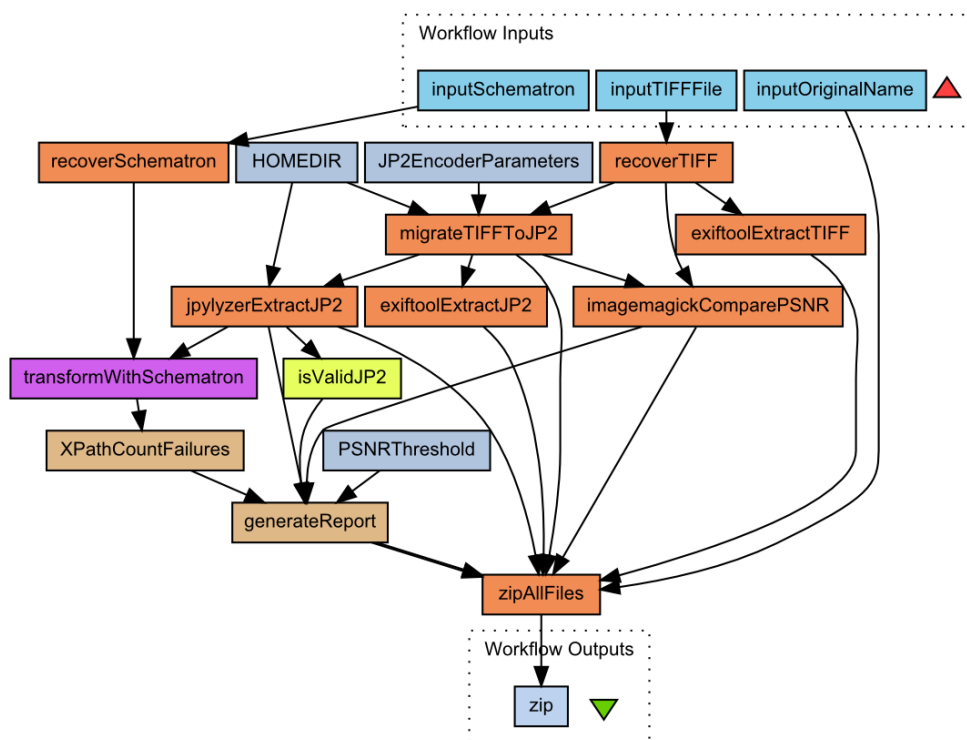


Figure 11 Example of a Taverna workflow that uses Jpylyzer

<b>Description of test dataset</b>	Metamorfoze sample batch, 8047 pages / TIFF images
<b>Link to test Dataset</b>	<a href="http://wiki.opf-labs.org/display/SP/KB+Metamorfoze+Migration+%28sample+batch%29">http://wiki.opf-labs.org/display/SP/KB+Metamorfoze+Migration+%28sample+batch%29</a>
<b>Code for Hadoop on GitHub</b>	<a href="https://github.com/KBNLresearch/hadoop-jp2-experiment">https://github.com/KBNLresearch/hadoop-jp2-experiment</a>
<b>Platform</b>	<a href="http://wiki.opf-labs.org/display/SP/KB+Hadoop+Platform">http://wiki.opf-labs.org/display/SP/KB+Hadoop+Platform</a>
<b>Number of nodes</b>	4 (1 master, 3 worker nodes)
<b>Total number of CPU-cores</b>	1
<b>CPU specs</b>	2.66 GHz Quad-Core
<b>Total amount of</b>	16

<b>RAM in GBytes</b>	
----------------------	--

Table 7 Description of dataset and execution environment for Metamorfoze Hadoop experiment

The workflow is made up of the following components:

- Use Exiftool<sup>47</sup> to extract metadata from the TIFF
- Migrate the TIFF to JP2 using the Aware JP2K SDK<sup>48</sup>
- Run Jpylyzer over the JP2
- Run Probatron<sup>49</sup> over Jpylyzer outputs to validate conformance of migrated image to the specified profile
- Use Kakadu<sup>50</sup> to convert the JP2 back to a (temporary) TIFF
- Use GraphicsMagick<sup>51</sup> to compare pixels of original and temporary TIFF.

Table 9 summarizes the performance indicators for each tool in the migration<sup>52</sup>:

<b>Tool</b>	<b>Time(h:m:s)</b>	<b>NumberOfObjectsPerHour</b>	<b>AverageRuntimePerItemInHours</b>
<b>Aware compress</b>	5:18:46	1518	6.59E-04
<b>Jpylyzer</b>	0:28:25	17244	5.80E-05
<b>Kakadu expand</b>	4:39:54	1731	5.78E-04

---

<sup>47</sup> <http://www.sno.phy.queensu.ca/~phil/exiftool/>

<sup>48</sup> <http://www.aware.com/imaging/jpeg2000sdk.html>

<sup>49</sup> <http://www.probatron.org/probatron4j.html>

<sup>50</sup> <http://www.kakadusoftware.com/>

<sup>51</sup> <http://www.graphicsmagick.org/>

<sup>52</sup> The raw evaluation results can be found here: <http://wiki.opf-labs.org/pages/viewpage.action?pageId=36012209>

<b>GraphicsMagick compare</b>	0:51:07	9467	1.06E-04
<b>Probatron</b>	3:38:18	2215	4.52E-04
<b>Overall</b>	14:56:33	539	1.86E-03

Table 9 Performance indicators of all tools in Hadoop workflow for Metamorfoze migration

Note that the figures in the *NumberOfObjectsPerHour* column represent the number of objects that each individual tool would be able to process in isolation. These results clearly demonstrate Jpylyzer's modest contribution to the overall processing time, which is equivalent to about 200ms per image. A more detailed description of this work can be found in Deliverable D 16.2.

The Austrian National Library (ONB) also did an experiment in which 1000 TIFF images were migrated to JP2. Overall the experiment is similar to the Metamorfoze one, although it is run on a real cluster, and it uses OpenJPEG for the actual migration. Table 10 gives an overview of the main characteristics of the dataset and the execution environment:

<b>Description of test dataset</b>	Austrian National Library Tresor Music Collection, 1000 TIFF images
<b>Link to test Dataset</b>	<a href="http://wiki.opf-labs.org/display/SP/Austrian+National+Library+Tresor+Music+Collection">http://wiki.opf-labs.org/display/SP/Austrian+National+Library+Tresor+Music+Collection</a>
<b>Taverna workflow</b>	<a href="http://www.myexperiment.org/workflows/4276.html">http://www.myexperiment.org/workflows/4276.html</a>
<b>Platform</b>	<a href="http://wiki.opf-labs.org/display/SP/ONB+Hadoop+Platform">http://wiki.opf-labs.org/display/SP/ONB+Hadoop+Platform</a>
<b>Number of nodes</b>	5
<b>Total number of CPU-cores</b>	20
<b>CPU specs</b>	Quad-Core
<b>Total amount of RAM in GBytes</b>	80

Table 10 Description of dataset and execution environment for ONB TIFF to JP2 migration experiment

Table 11 summarizes the performance of the individual processes in the workflow<sup>53</sup>. Figure 12 provides another view by displaying the total execution times of all processes.

Process	Time (h)	NumberOfObjectsPerHour	AverageRuntimePerItemInHours
<b>FITS validation</b>	1.17	857	1.17E-03
<b>migrate TIFF 2 JPEG2000</b>	5.22	191	5.22E-03
<b>Jpylyzer validation</b>	0.31	3273	3.06E-04
<b>re-migrate JPEG2000 2 TIFF</b>	5.36	187	5.36E-03
<b>compare pixel wise</b>	1.03	973	1.03E-03
<b>Overall</b>	13.08	76	1.31E-02

**Table 11 Performance indicators of all processes in ONB TIFF to JP2 migration experiment**

Although the absolute figures are understandably different to those of the Metamorfoze experiment, these results again highlight Jpylyzer’s small contribution to the overall processing time.

---

<sup>53</sup> The raw evaluation results can be found here: <http://wiki.opf-labs.org/display/SP/TIFF+to+JPEG2000+Migration+Experiment+at+ONB>

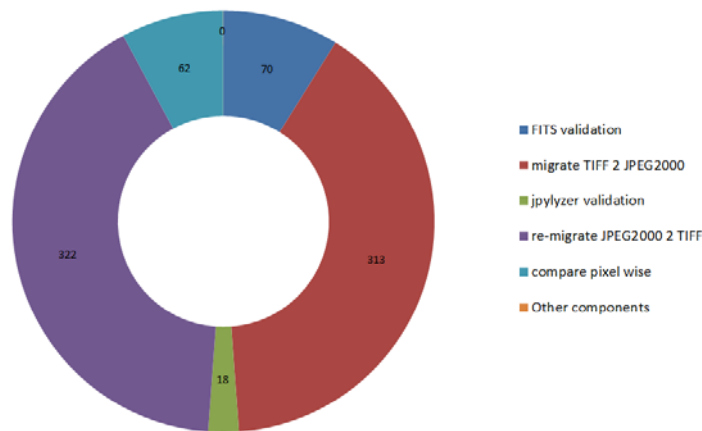


Figure 12 Total execution times (minutes) for all processes in ONB TIFF to JP2 migration experiment

## 5.2 Photohawk

Photohawk is a quality assurance tool for migration of born-digital images. This tool is related to SCAPE Scenario “LSDRT10 Capturing Representation Information from original image files”<sup>54</sup>, which has a task to run a Quality Assurance mechanism for migration of raw Canon images.

Photohawk is able to calculate the following list of comparison metrics:

- SSIM, structure similarity index
- rMSE, relative mean squared error
- rAE, relative absolute error
- rPAE, relative peak absolute error
- rMAE, relative mean absolute error



Figure 13 Photohawk logo

For a given pair of images and a preferred comparison metric, Photohawk produces a value from 0 to 1, where 0 stands for completely dissimilar and 1 for identical. The tool supports comparison of raw images, such as DNG, NEF, CR2 etc. This is done by converting raw images to a common TIFF format using dcrw, an open source tool used for raw image processing. Detailed description of Photohawk

<sup>54</sup> <http://wiki.opf->

[labs.org/display/SP/LSDRT10+Capturing+Representation+Information+from+original+image+files](http://labs.org/display/SP/LSDRT10+Capturing+Representation+Information+from+original+image+files)

is available in Kulmukhametov et. al. "Automated Quality Assurance for Migration of Born-digital Images"<sup>55</sup>.

### 5.2.1 Deployment

A detailed instruction for deployment and a usage guide is available at the repository. Besides usual java package, there is a Debian package and a Hadoop job available for Photohawk<sup>56</sup>.

### 5.2.2 Workflows

We have created Taverna workflows based on this tool. This enables reuse and experimentation with Photohawk and ensures its accessibility to the public. They are publicly available on myExperiment<sup>57</sup>. Figure 14 demonstrates one of the workflows. The workflow contains 3 nested sub-workflows. Such a workflow can be easily extended or modified, which increases applicability of the tool in different scenarios.

### 5.2.3 Experimentation

Experiments on correctness of the tool are described in the paper<sup>55</sup>. Here is described another experiment, which examines the dependency of processing time on the quantity of images. The test dataset was 230 raw images, taken from different cameras. Each original image was converted to DNG format using 3 different converters: Adobe DNG Converter, PhaseOne CaptureOne Converter and DxO Converter. This process resulted in 690 unique image pairs. Details on the dataset are given in the paper. To test the performance of Photohawk, Taverna server was used, running on the following configuration:

- CPU: Intel Xeon 8 cores @ 2.67GHz
- RAM: 64 Gb
- OS: Ubuntu Server 12.04
- HDD: 1 TB

The SSIM workflow was executed on the generated datasets. The results of experimentation are presented in Figure 14. It contains processing time for each pair of images and describes 2 processes:

---

<sup>55</sup> A. Kulmukhametov, M. Plangg and C. Becker. Automated Quality Assurance for Migration of Born-digital Images. In: Archiving 2014. Berlin, May (<http://www.ifs.tuwien.ac.at/~kulmukhametov/references/Archiving2014.pdf>)

<sup>56</sup> <https://github.com/datascience/photohawk>

<sup>57</sup> <http://www.myexperiment.org/packs/576.html>

SSIM calculation and conversion of raw image to TIFF. Index refers to each pair of images: original and a converted.

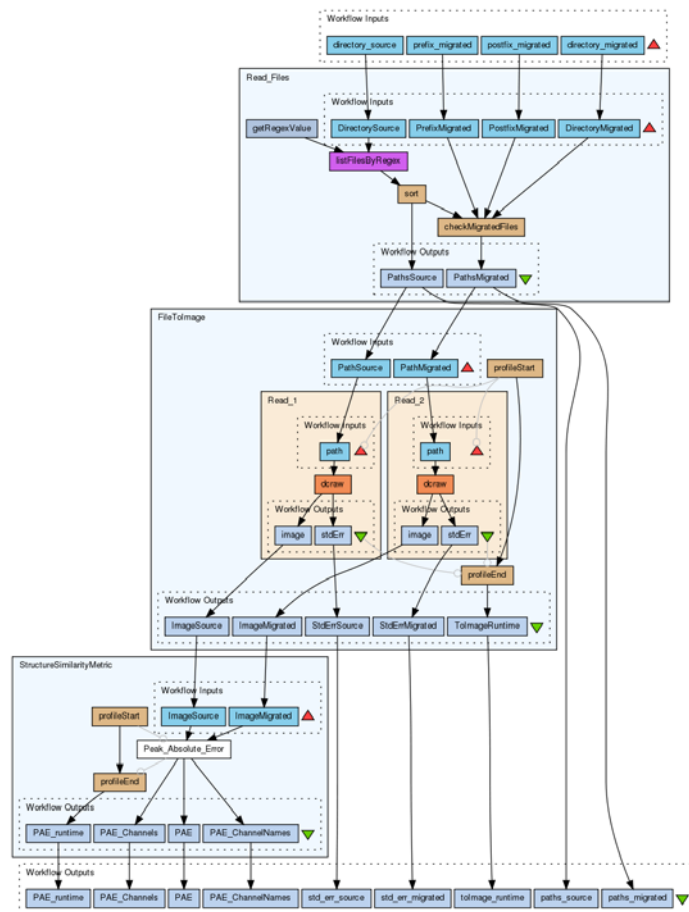
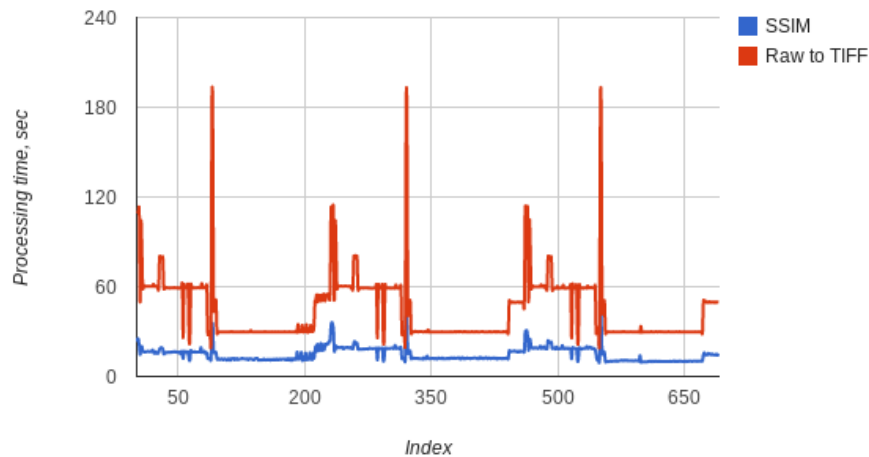


Figure 14 A Taverna workflow, based on Photohawk.

The SSIM workflow was executed on the generated datasets. The results of experimentation are presented in Figure 15. It contains processing time for each pair of images and describes 2 processes: SSIM calculation and conversion of raw image to TIFF. Index refers to each pair of the 690 images.





**Figure 15 Results of experimentation on performance**

As it could be seen from Figure 15, average time for calculating SSIM is 15s and average time for raw to TIFF conversion is 45s. This workflow composition provides a constant complexity of calculations over time. The workflow may be easily run in parallel on a bigger infrastructure with a linear dependency. Taverna server supports extension of a processing units list. Figures 16 and 17 contain results that support our claim. The Taverna workflow was executed in 2 modes:

1. In parallel to process 3 datasets, which were obtained by the converters. This required running 3 Taverna processes at the same time.
2. Sequentially, 1 Taverna instance is to process the full dataset.

The diagram in Figure 16 shows that it required approximately the same amount of time for each running Taverna instance to process its part of the data. Each Taverna instance is run in a separate thread, so there is extra time spent on process synchronization. The total processing time is about 13900 seconds.

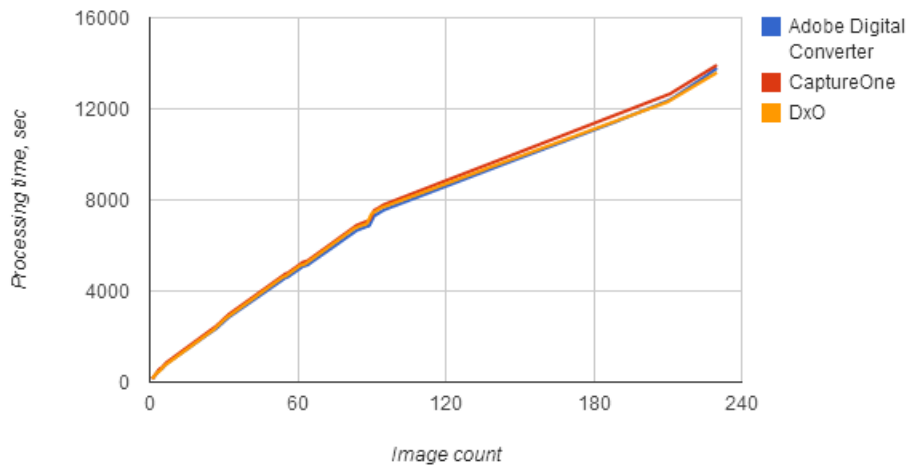


Figure 16 Results of running the SSIM workflow on the subsets on 3 instances of Taverna Workbench.

Figure 17 contains results of running the workflow on the whole dataset on one instance of Taverna. The total processing time is 41000 seconds. Studying the figures, we may conclude that the created workflows are of linear complexity. It is possible to split a dataset into parts and to run the workflows on separate instances for each of the subset, which may result in reduction of total processing time.

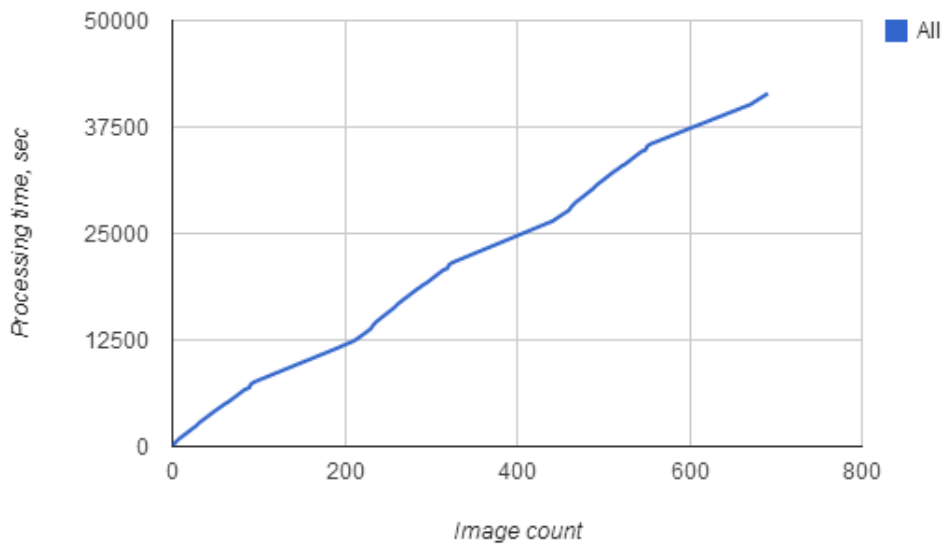


Figure 17 Results of running the SSIM workflow on the whole dataset on 1 instance of Taverna Workbench.

Experiments have been successfully run and they showed that Photohawk can run at scale. Results of experiments on correctness of this tool with detailed description of Photohawk and generated datasets are available in the paper by Kulmukhametov et al.

A toolspec that creates a debian package for easy distribution and a Hadoop job to enable the scalability is also provided<sup>58</sup>.

---

<sup>58</sup> <https://github.com/datascience/photohawk/tree/master/photohawk-commandline>

## 6 QA Supporting Tools

This section describes different QA tools used in different workflows to support overall QA work.

### 6.1 DRMLint<sup>59</sup>

DRMLint is a tool written in Java that was initially developed for PDF or EPUB files to perform:

1. *Format validation* – do the files conform to the format specifications?
2. *Feature extraction* – do the files contain DRM or encryption?

#### 6.1.1 Exemplar Use Case

The need for such a tool is borne out of the desire for policy-driven identification of preservation risks in electronic document formats<sup>60</sup>. Digital repositories typically hold large numbers of such documents from various sources, and include features that are potential risks for long-term preservation and accessibility, such as encryption, copy/edit/print protection, missing fonts, JavaScript, etc. The first step in mitigating these risks is being aware of which files contain them.

#### 6.1.2 Tool Development

Work began on DRMLint as no single tool provided a definitive answer to detecting DRM, and so DRMLint instead makes use of more than one software library to strengthen its results. The open-source PDF libraries available (particularly Apache PDFBox) are of a high quality, and rapidly developing features and robustness.

DRMLint is designed to reuse external libraries wherever possible, so as to maximize reuse of existing work. It contains an extensible design, where format specific tests are contained in a single location and new classes to cover many different formats can be easily added. In addition to making use of external libraries it contains internal logic for checking for DRM/encryption. The overall result is reported along with the results of all the individual checks that make up that result.

Work is ongoing to provide a GUI for the tool, to integrate the pdfPolicyValidate<sup>61</sup> work to assess fine grained features of PDF files, to enable institutional policies to be defined for individual formats, and to provide a SCAPE web demonstrator for the tool. Subsequently, the tool is going to be renamed

---

<sup>59</sup> Soon to be renamed Flint <https://github.com/openplanets/drmlint>

<sup>60</sup> <http://wiki.opf-labs.org/display/SP/Policy-Driven+Identification+of+Preservation+Risks+in+Electronic+Document+Formats>

<sup>61</sup> <https://github.com/openplanets/pdfPolicyValidate>

Flint, to better reflect the fact that its policy validation features have expanded beyond DRM and that its design means more formats can be easily added.

The API is documented with JavaDoc and a README file is provided containing details of how to build the software. Maven is used to make the build process easier. No public unit tests are currently available due to licensing issues with test files, although we hope to provide public unit tests in an upcoming update.

### 6.1.3 Testing at scale / Use in testbeds

DRMLint provides a built-in, straightforward MapReduce program that can be executed on a Hadoop cluster to demonstrate use of the tool at scale. A limitation of the demonstration MapReduce program is that files to have QA should be loaded directly into HDFS. This is a limitation of the MapReduce program, not DRMLint itself.

DRMLint has been tested at scale with the Large Scale Digital Repository testbed<sup>62</sup>. It was able to run over 231k files (127GB) in 4.5 hours on one Hadoop cluster.

Code for Hadoop on GitHub	<a href="https://github.com/openplanets/drmlint">https://github.com/openplanets/drmlint</a>
Description of test dataset	Govdocs1 Open Corpus: a corpus of 1 million documents that are freely available for research, drawn from US government web sites, of various formats. This dataset contains 231,683 PDFs which total 127.8GB
Link to test Dataset	<a href="http://digitalcorpora.org/corpora/files">http://digitalcorpora.org/corpora/files</a>
Number of nodes	1 master node, 28 worker nodes (1 map slot per node)
Total number of CPU-cores	1 CPU/node
CPU specs	AMD Opteron(TM) Processor 6276
Total amount of RAM in GBytes	6GB RAM/node
NumberOfObjectsPerHour	51869
Taverna Hadoop Workflow (if available)	NA

---

<sup>62</sup> <http://wiki.opf-labs.org/display/SP/EVAL-BL-LSDRT-PDFDRM-01>

## 6.2 Dissimilar<sup>63</sup>

Dissimilar is an experimental image quality assurance tool written in Java, formed of two main parts:

1. A Java library for calculating metrics of the differences between two images (PSNR<sup>64</sup> & SSIM<sup>65</sup>)
2. A graphical user interface (GUI) for loading pairs of images and comparing the two

### 6.2.1 Exemplar Use Case

Image migration requires processing of images from one format to another, along with associated quality assurance techniques to ensure the accuracy of this migration<sup>66</sup>. To enable large-scale image migration, all stages of this process need to consider performance, including the quality assurance tools. For example, one Experiment within the image migration User Story<sup>67</sup> performs image QA after a format migration by calling a native binary (ImageMagick), and this has an associated performance cost when used within a Java MapReduce program. By keeping image QA in native Java, it is hoped to improve performance.

### 6.2.2 Tool Development

The Java library contains an implementation of both PSNR<sup>68</sup> and SSIM<sup>69</sup> algorithms without any external dependencies. Initially, after a survey of available software, an attempt to reuse external libraries was attempted but without success – using the OpenCV<sup>70</sup> Java bindings added a dependency on external native-code binaries that needed to be installed on each Hadoop node, or bundled with the Jar, additionally some manual memory management was required within the Java code that made use of the library. These added complexities were not optimal to just use two algorithms. Therefore Dissimilar currently uses a separate library for image loading and runs the algorithms itself against the loaded image data.

---

<sup>63</sup> <https://github.com/bl-dpt/dissimilar>

<sup>64</sup> [http://en.wikipedia.org/wiki/Peak\\_signal-to-noise\\_ratio](http://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio)

<sup>65</sup> [http://en.wikipedia.org/wiki/Structural\\_similarity](http://en.wikipedia.org/wiki/Structural_similarity)

<sup>66</sup> <http://wiki.opf-labs.org/display/SP/Large+Scale+Image+Migration>

<sup>67</sup> <http://wiki.opf-labs.org/display/SP/LSVRT2+EX1+BL+Newspapers+on+the+BL+Platform>

<sup>68</sup> [http://en.wikipedia.org/wiki/Peak\\_signal-to-noise\\_ratio](http://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio)

<sup>69</sup> [http://en.wikipedia.org/wiki/Structural\\_similarity](http://en.wikipedia.org/wiki/Structural_similarity)

<sup>70</sup> <http://opencv.org/>

Unit tests, where the PSNR values are verified against the ImageMagick calculated values are contained within the repository. The code is commented and provides JavaDoc API documentation. No binary builds are available but it builds easily with Maven and build instructions are provided in a README file.

A GUI exists and can be used for manual assessment of pairs of images, allowing the user to assign a “pass” or “fail” to a pair of images. This could be used for generating a set of ground truth data for use when testing the library.

This work is similar in spirit to Photohawk (see Section 5.2), and was developed around the same time. Development of Dissimilar ceased approximately the time that Photohawk surfaced.

### 6.2.3 Testing at scale / Use in testbeds

Although Dissimilar has been integrated with code that can be run at large scale, no large scale evaluation has been performed. The current design of Dissimilar is RAM hungry and substantial changes to image loading would need to be made to increase performance and decrease RAM usage. Currently, Apache-Commons Imaging library is used to load images and a rewrite of Dissimilar to be more efficient would require reimplementing of a streaming read library for images. It is anticipated that this work would be involved and as such, and due to the existence of Photohawk, we have not attempted to implement it.

The code to run Dissimilar at large scale is in the Chutney-Hadoopwrapper<sup>71</sup> project, used for the image migration User Story in Large Scale Digital Repository Testbed<sup>72</sup>.

Code for Hadoop on GitHub	<a href="https://github.com/bl-dpt/dissimilar">https://github.com/bl-dpt/dissimilar</a> <a href="https://github.com/bl-dpt/chutney-hadoopwrapper/">https://github.com/bl-dpt/chutney-hadoopwrapper/</a>
Description of test dataset	19th Century Digitized Newspapers (2.2million pages of digitized 19th Century Newspapers)
Link to test Dataset	<a href="http://wiki.opf-labs.org/display/SP/BL+19th+Century+Digitized+Newspapers">http://wiki.opf-labs.org/display/SP/BL+19th+Century+Digitized+Newspapers</a>
Number of nodes	1 master node, 28 worker nodes (1 map slot per node)

<sup>71</sup> <https://github.com/bl-dpt/chutney-hadoopwrapper/>

<sup>72</sup> [https://github.com/bl-dpt/chutney-hadoopwrapper/blob/master/src/main/java/eu/scape\\_project/tb/chutney/jobs/CommandLineJob.java](https://github.com/bl-dpt/chutney-hadoopwrapper/blob/master/src/main/java/eu/scape_project/tb/chutney/jobs/CommandLineJob.java)

Total number of CPU-cores	1 CPU/node
CPU specs	AMD Opteron(TM) Processor 6276
Total amount of RAM in GBytes	6GB RAM/node
Large scale testing	No large scale testing carried out with this tool

### 6.3 JP2Check<sup>73</sup> library

The JP2Check Java library performs two main functions:

1. Using a standardized format profile to generate command line encoding parameters for multiple JPEG 2000 codecs (taking care of subtleties between them)
2. Processing the output from Jpylyzer using Schematron to validate the encoding profile of a JPEG 2000 image<sup>74</sup>

It was spun out of code that was developed for use in the Chutney-Hadoop wrapper testbed code. It is commented and provides JavaDoc API documentation. No binary builds are available but it builds easily with Maven.

There are some subtle differences in the ways in which parameters should be passed to JPEG 2000 codecs, notably the compression ratios and number of “levels” in the image. These are automatically handled by the library, enabling codecs to be interchanged quickly but still using the same Schematron validation for the same profile.

#### Testing at scale / Use in testbeds

The JP2Check library has been tested at scale in testbeds as part of an image format migration of 1TB TIFF images to JP2<sup>75</sup>. This workflow is described in D16.2 and also here: <http://wiki.opf-labs.org/display/SP/LSVRT2+EX1+BL+Newspapers+on+the+BL+Platform>. No separate testing of JP2Check as an independent library has been performed at large scale.

---

<sup>73</sup> <https://github.com/bl-dpt/jp2check>

<sup>74</sup> <https://github.com/bl-dpt/jp2check/blob/master/src/main/resources/jpylyzer-schematron.sch>

<sup>75</sup> [https://github.com/bl-dpt/chutney-hadoopwrapper/blob/master/src/main/java/eu/scape\\_project/tb/chutney/jobs/CommandLineJob.java#L609](https://github.com/bl-dpt/chutney-hadoopwrapper/blob/master/src/main/java/eu/scape_project/tb/chutney/jobs/CommandLineJob.java#L609)



Code for Hadoop on GitHub	<a href="https://github.com/bl-dpt/jp2check">https://github.com/bl-dpt/jp2check</a> <a href="https://github.com/bl-dpt/chutney-hadoopwrapper/">https://github.com/bl-dpt/chutney-hadoopwrapper/</a>
Description of test dataset	19th Century Digitised Newspapers (2.2million pages of digitised 19th Century Newspapers)
Link to test Dataset	<a href="http://wiki.opf-labs.org/display/SP/BL+19th+Century+Digitized+Newspapers">http://wiki.opf-labs.org/display/SP/BL+19th+Century+Digitized+Newspapers</a>
Number of nodes	1 master node, 28 worker nodes (1 map slot per node)
CPU specs	AMD Opteron(TM) Processor 6276
Total amount of RAM in GBytes	6GB RAM/node
Large scale testing	No large scale testing carried just on this tool, but it has been used within a larger test, see: <a href="http://wiki.opf-labs.org/display/SP/LSDRT2+EX1+BL+Newspapers+on+the+BL+Platform">http://wiki.opf-labs.org/display/SP/LSDRT2+EX1+BL+Newspapers+on+the+BL+Platform</a> and <a href="http://wiki.opf-labs.org/display/SP/EVAL-BL-LSDRT-TIFFJP2-01">http://wiki.opf-labs.org/display/SP/EVAL-BL-LSDRT-TIFFJP2-01</a>

## 7 Conclusion

According to the description of work, the workpackage is split into different tasks. However, during the kick off meeting held in Vienna, leaders of work package 11, along with sub-project Quality Assurance leader and project leader, decided to organize the work-package in 7 working groups oriented to media types. All the media types follow the same methodology. This organization has enabled a horizontal communication between all the work-package tasks.

The 6 tasks are listed as follows:

- ✓ Task 1 aims at producing quality assurance workflows that will integrate the tools developed in tasks 2, 3 and 4;
- ✓ Task 2 will output a range of tools capable of converting objects of various media types into a temporary format capable of being analysed and compared with another instance of that object, i.e. before and after any preservation action;
- ✓ Task 3 is intended to select and integrate appropriate analysis tools to measure the quality into the workflows defined in task 1;
- ✓ Task 4 provides new algorithms to quantifiably compare the converted objects;
- ✓ Task 5 will contribute to an open knowledge base to record, evaluate and share certain features, and how to detect if file objects are affected by them;
- ✓ Task 6 consists of validating and assessing conversion tools.

Through this last deliverable, the two previous ones (D11.1 and D11.2) and D11.4 Knowledge base of format-specific preservation risks (covering Task 5), we showed how we accomplished these tasks step by step.

PC.WP3 was able to, for each media type,

- ✓ Describe the issues with related scenarios
- ✓ Propose and implement new approaches,
- ✓ Provide proper deployment guides and deb packages
- ✓ Provide Taverna workflows
- ✓ Correctness based benchmarking based on ground truth
- ✓ Provide Hadoop implementation of tools
- ✓ Provide online demonstration pages

## Appendix A: Installation of Demo Web Sites

This section describes how to use VirtualBox and Vagrant to quickly set up a local VM that runs the SCAPE demonstrator for all tools <https://github.com/openplanets/scape-demo-sites/>.

An expert should install VirtualBox and Vagrant; these are available for Windows, OS X, and most common Linux distributions. The TCP/IP port 2020 should be open on your local machine.

First clone this repository and move into your local directory copy:

```
git clone git@github.com:openplanets/scape-demo-sites.git
cd scape-demo-sites
```

If this is the first time you're running the local demonstrator VM you'll need to install a Vagrant box, in this case an Ubuntu 12.04 LTS box. A box is a template image from which VMs are created by Vagrant. Issue the following command:

```
vagrant box add precise64 http://files.vagrantup.com/precise64.box
```

It may take a few minutes for the box to download. If you're unsure about whether you have the appropriate Vagrant box downloaded, issue the command:

```
vagrant box list
and look for the line:
precise64 (virtualbox)
```

amongst the listed boxes, on a Linux box `vagrant box list | grep precise64` can be used to thin the output if necessary.

After installing the box, from your local project directory issue the command `vagrant up`. This will start the headless VM. If this is the first time you have run the command, then it will provision the VM, that is install the appropriate software for the demonstrator. This is achieved by running the `bootstrap.sh` shell script. Open a browser and visit <http://localhost:2020> which should show the demonstrator home page.

To stop the demonstrator, from your local project directory issue the command `vagrant halt`. This will shut down the VM however the Vagrant box will continue to take up local resources, i.e. disk space, in the `.vagrant` sub-directory of the project. See Uninstall to see how to free up resources.

To restart the VM at any time issue the `vagrant up` command from the project directory.