USING THE IMAGERY OF BOOKS EXTRACTION TOOL V1.0
TECHNICAL DOCUMENTATION

Technical Contact
Kalev Leetaru [1]

ABOUT THE SOFTWARE

The Imagery of Books Extraction Tool is the software behind the 500 years of images of the world's books extracted from the Internet Archive's digital book holdings and made available on Flickr. [2] [3] [4] The tool is a simple 200 line PERL script that accepts as its input an Internet Archive Identifier and outputs a ZIP file containing the extracted images from the book in JPEG format along with a tab-delimited index file listing the details for each image.

The tool does not actually perform image analysis itself, instead relying on the book having already been processed using the commercial Abbyy OCR package [5] with XML output enabled. It works by parsing the Abbyy OCR XML file for the book, compiling a list of all image regions previously identified by Abbyy, and extracting those images to separate JPEG files using ImageMagick and Kakadu Software's image extraction libraries. Image region recognition is extremely computationally-intensive and requires incredibly sophisticated algorithms to correctly identify the boundaries of an image and to separate images from skew and other textual artifacts. The Abbyy software already has to identify all image regions as part of the OCR process so that it can avoid searching for text in those regions. By leveraging the work already done by Abbyy, the tool is able to rapidly process a given book to extract its images. While not perfect, the author is not aware of any software which has systematically improved upon Abbyy's image recognition accuracy by a substantial margin across a diverse array of materials and as improvements in the Abbyy OCR software are made, the image recognition process will continue to improve in the diversity of materials it can process.

The tool as currently written is designed for works digitized and hosted by the Internet Archive and makes use of a number of Archive-specific features such as its "scandata.xml" page type annotations and its server-based services-exposed ZIP decompression capability, which minimizes network and disk IO on the processing computer. It is anticipated that users wishing to apply this tool to their own digitized collections will use the existing PERL script as a template and modify it to their own needs and local system infrastructure. In particular, users processing large numbers of books may wish to replicate some of the technical functionality provided by the Internet Archive, especially a distributed server-based ZIP decompression facility exposed through a web service in order to absolutely minimize IO throughput on the processing computers (see the technical discussion at the end of this document).

---

[1] kalev.leetaru5@gmail.com / http://www.kalevleetaru.com/
[2] http://blog.archive.org/2014/08/29/millions-of-historic-images-posted-to-flickr/
[3] http://blog.flickr.net/en/2014/08/29/welcome-the-internet-archive-to-the-commons/
[4] http://www.bbc.com/news/technology-28976849
[5] http://www.abbyy.com/

Using the Tool

The entire image extraction system is contained in a single PERL script "fullresolutionimageextractor.pl." It requires the installation of ImageMagick's "convert" utility, [6] Kakadu Software's "kdu_buffered_expand" [7] utility and the CJPEG utility into the same directory. [8] It has currently only been tested under Linux and also requires the standard "wget" and "zip" utilities to be installed as well. It is run via "./fullresolutionimageextractor.pl IDENTIFIER", where IDENTIFIER is the Internet Archive Identifier of the desired book. It will then process the book and output a ZIP file into the ZIPCACHE directory when it is done.

The basic workflow is as follows:

- **Download File List**. The list of all files available for this book is downloaded and checked to ensure that the book has available a valid Abbyy OCR XML file, a "scandata.xml" file listing which pages should be skipped (such as color bar pages, etc), and a ZIP file containing the original camera imagery for the book. If any of these are missing the system exits.
- **Download Abbyy OCR XML File**. The Abbyy OCR XML file is downloaded to local disk and uncompressed.
- **Download scandata.xml File**. The Internet Archive-specific "scandata.xml" file is downloaded that contains a list of all of the camera page images, their type/status and the page number offset (whether the first page begins as page 0 or page 1). During the scanning process "non display" images such as color cards, rulers, or blurred camera images are not discarded and are processed by the Abbyy software, but are excluded from the page numbering for the final human-displayed book – the "scandata.xml" file identifies these pages so they the page numbering of the Abbyy OCR XML file can be translated into the final human-displayed page numbering to align with the Archive's online book viewer. The code in this section should be adjusted to integrate the page offsets and page type/status files used at your site.
- **Process Abbyy OCR XML File**. The Abbyy OCR XML file is processed to identify all image regions. NOTE - images in the first three or last three pages are discarded, since these were found to almost exclusively contain cover inset decorations or other imagery unrelated to the content of the book. Images of less than 300x300 pixels are also discarded – in practice these were found to be almost exclusively light page damage in the margins or spine area of the page. Note that on some occasions Abbyy can incorrectly segment a single image into two images in the case where two high detail regions are separated by a small low-detail connector. Additional code could be added here to connect image regions that overlap. Finally, if the book had less than three pages containing images or less than four separate images (since a page could contain multiple images) it is discarded. This can result in some books, in particular from the late nineteenth century, that contain only a single title page image, being incorrectly discarded. However, in practice over the entire collection this was determined to dramatically reduce the number of false positives from books that do not contain content imagery, but which contained artifacts, such as heavy physical damage, on one or two pages at the start of the book, which resulted in false positive image extractions from those pages.
- **Camera Image Fetch Decision**. At this point, if the book has more than 50 pages containing images to be extracted, the system downloads the complete camera imagery ZIP file to local

[6] http://www.imagemagick.org/
[7] http://www.kakadusoftware.com/
[8] http://jpegclub.org/cjpeg/

disk. If there are less than 50 pages containing imagery, it will download only the pages it needs as it processes the book. The cutoff of 50 pages was determined experimentally over hundreds of thousands of books as the optimal threshold on the Internet Archive's computing systems – you might adjust this on your own systems. The goal here is to absolutely minimize the amount of data that must be downloaded to or read from the local disk. The vast majority of books have their imagery concentrated on a few tens of pages, making it possible to just download those pages, rather than the entire book of several hundred pages. Once there are more than 50 pages that must be processed, it is faster to download the entire ZIP file locally.

- **Extract Surrounding Text**. The system processes the Abbyy OCR XML file a second time to rewrite and simply it for the final pass that extracts the text surrounding each image.
- **Generate the Command File**. The system processes the Abbyy OCR XML file one final time to extract the text surrounding each image and write a UNIX shell file filled with the necessary commands to extract the images from the page and write the final ZIP file. No actual processing occurs during this time, the tool only writes the final .SH shell file that will execute all of the necessary actions. The general structure is that if the full camera image ZIP file had been previously downloaded, the specific requested pages are unzipped individually to minimize disk IO – otherwise the requested pages are fetched via the Archive's services-based unzip functionality, downloading only the requested pages. Note the use of "kdu_buffered_expand" rather than the more frequently used "kdu_extract" – the former offers integer-based subregion extraction, allowing the cropping to occur simultaneously to conversion. Also note that multithreading is disabled due to an odd race condition that can cause corruption under high load.
- **Cleanup and Execute the Command File.** Finally, the system cleans up and exits, executing the shell file it previously created as it exits. The reason for separating the primary tool from the actual image extraction tasks is that Abbyy OCR XML files can be extremely large and many cloud-based virtual machines and HPC clusters are configured with very little memory per core, meaning that if the core PERL script, with the Abbyy OCR XML file loaded in memory, attempts to run at the same time as the image extraction tools, which must load a high-resolution JPEG or JP2 file into memory, systems often run out of available memory. Separating the two tasks minimizes the memory footprint.

Final Output

The final output of the tool is a ZIP file named with the IA Identifier of the book. Within the ZIP file is a JPEG file for each extracted image and a tab-delimited file containing the information about each image containing the fields below. Each image appears as a single row in the file and rows are separated by carriage returns. Images less than 30K in filesize should ordinarily be discarded (see the following section).

- **Identifier**. This is the Internet Archive's unique identifier for the book this image was extracted from.
- **Page Number**. The page number in the book that the image appeared on. Note that this is not the same as the page number of the raw page scan image that the image was extracted from (due to deleted pages flagged in "scandata.xml"), nor is it the actual printed page number, but rather corresponds to the page numbering used by the Archive's online interactive book viewer application for this book.

- **Image Number**. Each image extracted from the book is numbered sequentially from 0 to the total number of images extracted from the book. There may be gaps in this sequence if an image was extracted but later discarded as too small.
- **Width**. The width of the image in pixels.
- **Height**. The height of the image in pixels.
- **ImageFileName**. The filename of the image as stored in the ZIP file. This is of the format "Identifier.ImageNumber.PageNumber.jpg". Regardless of the file format of the original source page scans, all extracted images are saved as JPEG files. PageNumber is left-zero-padded to ensure it is always four digits long.
- **Filesize**. The size of the extracted image in bytes.
- **PageAccessURL**. This URL displays the book in the Internet Archive's online book viewer application, opened to the page containing this image. Note that in a small number of books a complication with the page numbering means that this may occasionally open to the page immediately before or after the image, instead of the page containing the image, but this is rare.
- **ImageAccessURL**. This URL uses the Internet Archive's "ZIP View" API to return this image, allowing an individual image to be requested without having to download the complete ZIP file of all images extracted from this book. This places a substantial load on the Internet Archive's servers so should only be used for experimentation and not for production applications.
- **PreText**. This contains the 1,000 characters of text that immediately precede the image on the page. If an image appears near the start of a page, this will include text from the previous page(s) up to the 1,000 character limit. If another image appears less than 1,000 characters before this image, the text is truncated to include only the text appearing after the previous image.
- **PostText**. This contains the 1,000 characters of text that immediately follows the image on the page. If an image appears near the end of a page, this will include text from the following page(s) up to the 1,000 character limit. If another image appears less than 1,000 characters after this image, the text is truncated to include only the text appearing before the following image.

Excluding Images

As a general rule of thumb, the following filters should be applied to the extracted images to minimize the inclusion of noise imagery and return the highest quality material. While these filters may result in some legitimate imagery being discarded, overall they eliminate the majority of false positives.

- **Small Pixel Size**. Skip images less than 300 x 300 pixels. This may inadvertently eliminate some smaller images, especially some less detailed "inhabited initials," but in general this eliminates a large number of noise images such as marks or tears, stamps, margin notes, and other artifacts that are not part of the original imagery of the book.
- **Small File Size**. Due to the way the JPEG compression algorithm works, the file size of a JPEG image offers a coarse approximation of its "visual complexity." Images less than 30K in size should be skipped, as they are usually simplistic images, such as a tear in a page or an ink splotch that may be larger than 300 x 300 pixels and thus still pass the pixel size filter above.
- **Extreme Aspect Ratios**. Images with significant aspect ratios should be excluded. In general, images in which width/height is less than or equal to 0.3 or height/width is less than or equal to 0.21 should be excluded. These usually indicate that the original page scan was not properly

cropped before being provided to the Abbyy OCR software such that the page borders were improperly recognized as images.

- **First and Last Images**. The "scandata.xml" file for each book usually indicates the locations of the front and back covers, but not the decorative interior pages at the start and end of the book. Especially in the late nineteenth century it was common to inlay elaborate designs on the first two pages immediately after the cover (before the title page) and the last two pages. Many libraries would additionally include bookplates or even tiled bookplates on these pages, while books from the late nineteenth and throughout the majority of the twentieth would include an envelope on these pages for the book's checkout card. Thus, the majority of the images from the first four pages of a book and the last five pages are noise images and should be discarded. Occasionally this will lead to legitimate images being discarded from books in which these pages were skipped by the human scanner or were rebound to exclude these pages, but in general for the vast majority of books this greatly reduces the number of noise images.
- **Too Few Images**. Finally, after the above filters have been run, the final number of remaining images and pages containing images should be examined. If the book contains less than 4 images or less than 3 pages with images (regardless of the number of images on those pages) the entire book and all of its remaining images should be discarded. This does eliminate some small number of books that feature only a single portrait image at the start of the book, but in return eliminates a vast number of books whose only images are assorted marginalia and/or library artifacts. After extensive testing the vast majority of books eliminated through this process contained only noise images and thus this final filter significantly reduces the last set of noise images to make it through the above filters.

Important Nuances and Caveats

With a collection as large as the Internet Archive's entire digital book corpus, and with as many organizations scanning those books over such a long time period, there are invariably imperfections, oddities, and nuances to the images resulting from this process. Although the filters above will minimize many of these issues, they should still be kept in mind when working with the collection.

- **Library Artifacts as Images**. Mixed among the illustrations, photographs, maps, and other visual elements of the books, you will find an assortment of library stamps, hand-drawn margin notes, torn and/or taped pages, and other artifacts stemming both from the book's ordinary wear and tear and from various processes it has undergone over the years as a library item. For example, some libraries would formerly use an ink stamp to mark random pages throughout a book (perhaps to deter theft) – each of these stamps will be extracted as an image. Some books contain significant amounts of drawing, underlying, and notes written in the margins by readers over the years, each of which are extracted as separate images. To minimize the number of these artifact images, only images larger than 300x300 pixels are extracted, but there are still a fair number of these artifacts that can make it through. Many books also feature bar codes, library card envelopes, book plates, and other devices in the first few pages and last few pages. It is highly recommended that images appearing in the first and last pages of a book be skipped.
- **Scanning Artifacts as Images**. In addition to actual artifacts on the pages of each book, there are also artifacts introduced through the scanning process. Each book is scanned with one or more "color cards" which are essentially a piece of paper with a series of color bars that are used later on to calibrate the color temperature of the scan. These color cards are supposed to be identified by the human scanner operator when the book is scanned and flagged for removal,

but occasionally they slip through. In addition, you may see images of the edge of a book with what appears to be black non-slip kitchen cabinet liner (a bumpy black foam-looking material). These are images that captured part of the pad that the book rests on when it is being scanned and were not correctly flagged by the human scanner operator.

- **Blank/Corrupt Images**. Periodically you may come across an image that is either blank or corrupt. This can occur for a variety of reasons, but usually is a result of one of the original page scan images being corrupt in a way that Kakadu's "kdu_buffered_expand" or ImageMagick's "convert" utility were unable to catch. The best way to detect these is to check for error return codes from the JPEG library you are using. Such images should be exceedingly rare.
- **Wrong Page Number**. A small number of books appear to have a mismatch in the page numbering between the raw page scans, the Abbyy OCR XML file, and the scandata.xml file such that the URL provided for each image to view it in context on the Archive's website using the online interactive book viewer may be one to two pages off. These are fairly rare.
- **Blurred Images**. Sometimes you will find images that are blurred or smeared. These are cases where the original page scan was not performed properly and the page moved slightly while it was being photographed. Sometimes this is not visible until you zoom into the page and look at the fine detail.
- **Poor Quality PreText/PostText**. The quality of the PreText and PostText fields varies dramatically depending on the age of the material, its condition, its language, the positioning of images and text on the page, and whether images appear throughout the text or are bunched together at the end of the book. Always remember that this is the original raw "as is" OCR results and so can be highly noisy. Also keep in mind that word usage, spelling, and grammatical standards have changed dramatically over the past 500 years.

THE "MAKING OF" A 500-YEAR ARCHIVE: BEHIND THE SCENES TECHNICAL WORKFLOW

The following section will be of interest to more technical users of the system to understand the technical specifics of how it was created and the key design decisions behind its core components. Those attempting to build their own mass extraction workflows will likely find the following section of particular importance and utility.

The Original Prototype

Like any large project, this one started off with an experimental prototype using low-resolution images to demonstrate both that processing the Archive's entire digital book collection was tractable and that the resulting image gallery would contain a wide array of visually interesting imagery.

So, how exactly does one go about creating an archive of images spanning 500 years? It all begins with the ebook. Historically, PDF versions of digitized books were created as what is called "image over text" files in which the scanned image of each page is displayed with the OCR'd text hidden underneath. This works well for desktops and laptops with their unlimited storage and bandwidth, but can easily yield files in the tens or hundreds of megabytes. Ereader devices, with their limited storage capacity and bandwidth, necessitated the adoption of more optimized file formats that extract the images from each page and save a book as ASCII text with embedded images, much in the way a web page is created. With the rise of the ereader, many digital libraries, including the Internet Archive, now make their books available in the open EPUB file format, which is essentially a ZIP file containing a set of HTML pages and

the book's images, each extracted as a separate image file (usually JPEG, PNG or GIF). Extracting the images from each book is therefore as simple as unzipping its EPUB file, saving the images to disk, and searching the HTML pages to locate where each image appears in the book to extract the text surrounding it.

The simplicity of this process is what makes it so powerful. The hardest part of creating an image gallery from books lies in the image recognition process needed to identify and extract each image from the page, yet this task is already performed in the creation of the EPUB files. By simply reusing the EPUB files in a creative new way one can unlock the visual dimension of millions of books with just a few lines of code. It also means that this process can be easily repeated for any digital library that offers EPUB versions of their works, making it possible to one day create a single master repository of every image published in every book ever digitized. The entire processing pipeline of all two million Internet Archive books was performed on a single four-processor virtual machine in the Internet Archive's new Virtual Reading Room in just a few days. [9] In this case, while all of the books used are available for public download on the Archive's website, using the Virtual Reading Room made it possible to work much more easily with the Archive's collections, dramatically reducing the time it took to complete the project.

A New Resolution: Creating the Final Archive

The EPUB prototype demonstrated the incredible variety of imagery and the value of extracting the text surrounding each image. However, since EPUB files are designed for portable reading devices with small screens and highly limited storage and network capabilities, the images in EPUB files are very low resolution, typically around 200x200 to 700x700 pixels maximum. This eliminates the majority of the fine resolution of most images and renders them unsuitable for anything other than casual inspection. Extracting the images at higher resolution requires going back to the original raw page scans of each page directly from the book scanning process.

The final system does precisely this, coupling the original full resolution page scan imagery with the full XML output of the Abbyy OCR system [10] and the Internet Archive-created "scandata.xml" information that records which pages are non-displayable like color calibration frames or bad captures. Together, this information is combined to extract each image at the maximum resolution the book was originally scanned at.

The extraction system begins by downloading the master file list for the requested book that displays all available files for the book including source (original page scan images, Abbyy OCR file, etc) and derivative (EPUB, PDF, etc) files. The URL is simply "http://archive.org/download/" followed by the book's identifier, such as "http://archive.org/download/liltobirds00pick". This returns an HTML file that is then parsed to locate the Abbyy OCR XML file, the original page scan imagery (JPEG2000, JPEG, or TIFF format), and the "scandata.xml" file. If any of these are missing, the book is skipped. Otherwise, the scandata.xml and the Abbyy OCR XML file are both downloaded to local disk. Books that use the older scandata.zip format are also skipped due to significant differences in how page and pixel offsets were calculated in the older format. While in theory all files for a book should follow the file naming schema

[9] http://www.knightfoundation.org/blogs/knightblog/2014/1/7/internet-archives-virtual-reading-room-empowers-data-mining-societal-scale/
[10] http://www.abbyy.com/

of “identifier.extension” (such as “identifier.pdf” or “identifier_j2.zip”), in reality many do not, and thus the system renames all files to the book’s unique identifier to standardize workflow.

Next, the “scandata.xml” file is examined to locate page scans that were flagged by the human operator of the book scanning system for exclusion. An example might be a bad page scan where the page was not in proper position for scanning and thus was rescanned. In this case instead of deleting the bad scan image, it is simply flagged in the scandata.xml file. Similarly, to ensure proper color calibration, a “color card” and ruler is photographed before and after each book is scanned (and sometimes periodically at random intervals throughout). These frames are also flagged in the scandata.xml file for exclusion. These page scans were dropped before the Abbyy OCR software was run on the book and thus must be identified to match the page numbering of the OCR XML file with the actual page scan images. For example, the first image in a book scan (image 0) might be the color card, and the second and third images might have been bad scans. Thus, in the ZIP file containing the page scan images, image0.jpg, image1.jpg and image2.jpg would all be excluded before Abbyy was run on the book to OCR it, meaning that “page 0” in the Abbyy file actually corresponds to “image3.jpg” in the page images ZIP file. Skipped pages can also occur in the middle of a book. If page 100 of this book slipped while the camera was photographing it and became blurred, the scanner operator might rescan that page and flag the original scan image as bad. Combined with the three skipped pages at the beginning of the book, this means that page 100 of Abbyy OCR file would actually correspond to image104.jpg in the page scan ZIP file (100 + the 3 skipped images at the start + 1 image for the original bad scan). The extraction system thus uses the “scandata.xml” file to compute this mapping between Abbyy OCR page numbers and the actual page scan image numbering for each book.

Now, the Abbyy OCR XML file is examined. This file contains an enormous wealth of information normally invisible to users of the consumer version of the Abbyy OCR software. It breaks up each page into a series of paragraphs, each paragraph into lines, and each line into individual characters. It even provides a confidence measure on how “sure” it is of each individual letter. The following is a sample excerpt that shows the format of the Abbyy XML stream capturing part of the word “digitize” followed by an image (the image appeared later in the page and the XML in between is truncated for clarity). Each region, line, character, and image includes “l,t,r,b” parameters are are its “left”, “top”, “right”, and “bottom” coordinates, in pixels, within the original page image. The first character, “d” has the XML tag “<charParams l="451" t="1491" r="481" b="1530" wordStart="true" wordFromDictionary="true" wordNormal="true" wordNumeric="false" wordIdentifier="false" charConfidence="100" serifProbability="0" wordPenalty="0" meanStrokeWidth="52">D</charParams>,” which indicates that it is located at a position 451 pixels from the left of the page and 481 pixels from the top of the page and is 481-451 = 30 pixels width by 1530-1491 = 39 pixels high. To extract the actual image of this character on the page, you would simply crop this region from the original page scan image. Images similarly appear with the simple tag “<block blockType="Picture" l="50" t="2" r="1202" b="2420">”, indicating an image that is located 50 pixels from the left of the page and 2 pixels from the top and is 1202-50 = 1,152 pixels width by 2420-2 = 2,418 pixels high.

<page width="1649" height="3172" resolution="500" originalCoords="true">
<block blockType="Text" l="436" t="1480" r="1220" b="1688">
<region><rect l="436" t="1480" r="1220" b="1688"></rect></region>
<text>
<par align="Center" lineSpacing="70">
<line baseline="1531" l="451" t="1491" r="1201" b="1542"><formatting
lang="EnglishUnitedStates" ff="Arial" fs="8." spacing="-3"><charParams l="451" t="1491" r="481"
b="1530" wordStart="true" wordFromDictionary="true" wordNormal="true" wordNumeric="false"

wordIdentifier="false" charConfidence="100" serifProbability="0" wordPenalty="0"
meanStrokeWidth="52">D</charParams><charParams l="489" t="1491" r="494" b="1530"
wordStart="false" wordFromDictionary="true" wordNormal="true" wordNumeric="false"
wordIdentifier="false" charConfidence="100" serifProbability="255" wordPenalty="0"
meanStrokeWidth="52">i</charParams><charParams l="499" t="1501" r="524" b="1542"
wordStart="false" wordFromDictionary="true" wordNormal="true" wordNumeric="false"
wordIdentifier="false" charConfidence="100" serifProbability="19" wordPenalty="0" meanS
trokeWidth="52">g</charParams><charParams l="531" t="1491" r="536" b="1530"
wordStart="false" wordFromDictionary="true" wordNormal="true" wordNumeric="false"
wordIdentifier="false" charConfidence="100" serifProbability="255" wordPenalty="0"
meanStrokeWidth="52">i</charParams><charParams l="540" t="1494" r="553" b="1531"
wordStart="false" wordFromDictionary="true" wordNormal="true" wordNumeric="false"
wordIdentifier="false" charConfidence="46" serifProbability="28" wordPenalty="0"
meanStrokeWidth="52">t</charParams><charParams l="558" t="1491" r="563" b="1530"
wordStart="false" wordFromDictionary="true" wordNormal="true" wordNumeric="false"
wordIdentifier="false" charConfidence="100" serifProbability="255" wordPenalty="0"
meanStrokeWidth="52">i</charParams><charParams l="568" t="1502" r="591" b="1530"
wordStart="false" wordFromDictionary="true" wordNormal="true" wordNumeric="false"
wordIdentifier="false" charConfidence="100" serifProbability="255" wordPenalty="0"
meanStrokeWidth="52">z</charParams>
…
<block blockType="Picture" l="50" t="2" r="1202" b="2420">

Thus, one simply has to scan the Abbyy XML file for all tags like “<block blockType="Picture" l="50" t="2" r="1202" b="2420">” to locate all images recognized by Abbyy in the book. Then it is a trivial matter of cropping them from the original page scan images and compiling the text that Abbyy located before and after each image.

Now that the list of images and their surrounding text has been compiled, the system must download the full-resolution page scans to extract the images from. The problem is that the page scans for each book are delivered as ZIP files that are usually several hundred megabytes, occasionally exceeding one gigabyte. While at first this may not seem like much in an era of 4TB USB drives, when multiplied by a large number of books, the network bandwidth and the speed of computer harddrives become critical limiting factors. For example, a set of 22-core virtual machines were used to handle much of the computing needs of this project. Typically one might attempt to process one book per core, meaning 22 books being processed in parallel at any given time. If all 22 books had 500MB ZIP files containing their full resolution page scan imagery, this would require downloading 11GB of data over the network. Assuming that each book takes less than a second of CPU time on average to process, this would require a 100Gbps network link working at 100% capacity to sustain these processing needs. Even if this was broken into 22 separate virtual machines, each with a single core, all 22 machines would still each require their own 4Gbps network link working at 100% capacity and delivering 500MB/s bandwidth. This, of course, is assuming a perfect 100Gbps sustained stream between the cluster of computers and the Internet Archive, which is highly unlikely over distance.

Even if the network bandwidth limitations are overcome, the greatest challenge is actually the disk bandwidth of writing a 500MB ZIP file to disk from the network and then unpacking it (reading the 500MB) and updating the file system metadata to handle several hundred to several thousand new files being written to disk (writing its 500MB of contents to disk). Thus, all said and done, a single 500MB ZIP file requires 500MB to be read twice and written twice, totaling 2GB of total IO. Processing 22 files per second would require 44GB/s of disk bandwidth. While the reads would likely come from kernel buffer cache, the 2GB of write IO has to pass through to disk. Many cloud computing vendors limit virtual

machines to around 100-120MB/s sustained writes and 180-200MB/s sustained read performance, while even a dedicated physical 3Gbps SATA harddrive operating at peak capability would still require 2 seconds just to write the ZIP file to disk and another two seconds to unpack it and write its contents to disk as individual files (assuming the intermediate reads are kernel buffered). Since writes are linear, SSD disks do not provide a speed advantage over traditional mechanical disks. Carving a portion of RAM into a virtual file system (a "RAM disk") would alleviate some of these challenges, but requires substantially larger amounts of memory and more elaborate management of the system. Ultimately, sustaining the kind of disk IO required of such a project requires local hardware RAID to spread the IO requirements over a large number of disks. While consumer RAID units are capable of 400-800MB/s read bandwidth, that assumes single linear file IO, whereas here 22 files must be processed in parallel, exponentially decreasing the available IO bandwidth as the disks begin thrashing. Even if 22 separate single-core VMs are used to overcome network bandwidth limits, this still requires each VM to have a disk system capable of 2GB/s bandwidth, which is quite rare today. Even the high performance parallel disk systems used on HPC systems degrade exponentially under this kind of high-intensity IO spread across a large number of files. Specialized filesystems designed for data-intensive processing require considerable dedicated resources and setup, rendering them impractical for this kind of project.

Instead, the processing pipeline was redesigned to minimize network and disk IO at all costs, even if it means slowing down the processing of a single book, if that allows more books to be processed in parallel (maximizing throughput). In the final system (developed after extensive experimental benchmarking), if a book has less than 50 pages containing images to be extracted, the page scan ZIP file is not downloaded and instead the Internet Archive's "ZIP Viewer" API is used to extract the needed page scan images individually via calls to the Archive's web service, at an average delay of around 2 seconds per image. A book with 50 pages containing images would therefore require around 1.6 minutes to download all of the images, whereas on the virtual machines used for this project the raw page scan ZIP could be downloaded in under 30 seconds. However, downloading multiple full ZIP files would exceed the network and disk bandwidth on the virtual machines, raising the total time required to download the full ZIP file to 5-15 minutes, meaning that when processing large numbers of books, downloading the individual page images directly from the API is considerably faster. For those books with more than 50 pages worth of images, the full ZIP is downloaded, but only the needed pages are unpacked from the ZIP file instead of all pages, dramatically reducing the read/write bandwidth beyond the original write from the network. Combined, these two techniques reduced the IO load on each virtual machine to the point that the CPUs were able to be kept largely occupied when running 44 books in parallel (twice the number of physical cores, overlapping the latency for the books being fetched through the page-level API).

Once all of the necessary full resolution page scan images are present on disk, the Abbyy-recognized images must be extracted via an image crop operation. Full resolution camera images are stored in either JPEG or JPEG2000 format. JPEG files are handled via ImageMagick's [11] "convert" utility, but JPEG2000 support in ImageMagick is too slow to work at this scale, requiring up to 8 seconds per image to read the JPEG2000 file, crop the image from it, and write the extracted image back to disk in JPEG format. Instead, Kakadu Software's [12] "kdu_expand" tool is used to extract the image from the JPEG2000 file, while the CJPEG [13] utility is used to write the resulting image stream to disk as a JPEG file. In this case a variant of kdu_expand known as "kdu_buffered_expand" is used, which uncompresses

[11] http://www.imagemagick.org/
[12] http://www.kakadusoftware.com/
[13] http://jpegclub.org/cjpeg/

only the specified subregion region of the JPEG2000 image instead of uncompressing the entire image and then cropping the requested region from it. This resulted in a speedup of between 8 and 30 times compared with ImageMagick, depending on the image. To minimize memory requirements, the final system actually generates a shell script and then exits and invokes the shell script to perform all of the actual image processing components of the pipeline, freeing up the memory originally used to process the scandata.xml and Abbyy OCR XML files since per-core memory was highly limited on the systems available for this project. Finally, the list of extracted JPEG images and a tab-delimited inventory text file listing the attributes of each image and the text surrounding them is ZIP'd up into one ZIP file per book, ready for use.

Thus, while conceptually quite simple (compile a list of images from an XML file, compute their page numbers using another XML file, and crop them out of the original page scans), the final system required considerable iterative development and enormous effort to minimize IO at all costs. This is one of the challenges of the new computing paradigm of the data-intensive world. In the past most large-scale computing involved computationally-bound problems that had little network or disk accesses and largely were entirely CPU-driven, such as simulation. Increasingly in the "big data" world processors are idled because data can't move fast enough to keep them occupied.