Scaling Document Preservation: Ancestry.com's Approach to Rapidly Digitizing Diverse Content

Joshua B. Harman; Ancestry.com, Inc.; Provo, Utah/USA

Abstract

Ancestry.com ingests and processes tens of millions of images per year. These images are captured from physical media such as loose paper, microfilm, and microfiche, and obtained from digital sources. This paper outlines how Ancestry.com has developed a system of technologies which allow the import images from diverse sources -- in many formats and from dozens of partners-- and produce a standard output.

The outcome of these technologies and platforms allows rapid production of content with configurable and scalable settings, easy adjustments to new workflow and audit requirements, ingestion of new image types with minimal cost, and delivery to partner and website requirements within a single system.

At Ancestry.com we believe our philosophy is revealed in full by the way we solve our most difficult operational problems

Introduction

Ancestry.com has been involved in process of digitizing, indexing, and hosting historical records since 1996. During that time, more than 4 petabytes of digital historical content have been made available on Ancestry.com.

As one of the pioneers of online family history research, Ancestry.com has developed systems capable of capturing, processing, and delivering tens of millions of new images per year at an increasingly rapid rate. Many of the updates and upgrades to Ancestry.com's systems have allowed breakthroughs which have been non-linear.

One of the keys to translating large amounts of historical content into something digital is found in developing systems and processes capable of handling highly variable content and very large amounts of data. While the acquisition and digitization of historical content is the first and most important step in this process, operational systems and methods for handling this content cannot be overlooked.

The purpose of this paper is to give an overview of the systems, processes, philosophies, and best practices which allow Ancestry.com's Document Preservation Services (DPS) department to ingest, process, and export hundreds of unique collections every year.

Document Digitization Background

For the more than 15 years Ancestry.com has been involved in the pursuit of digitizing historical content, many different methods and techniques have been employed to maximize speed, quality, and scalability. Initially, ingesting and processing a few dozen content collections per year was standard. However, as Ancestry.com accelerated the rate at which it has delivered content to users this number has raised into hundreds of collections per year. This presented a large problem as image processing often occurred on the same machines with which the images were scanned and both the hardware and process were difficult to scale.

In addition to accelerated production rates, more partnerships were created often adding complex new processing and delivery requirements. Managing so many projects simultaneously with such diverse specifications grew increasingly difficult.

The system lacked an adjustable workflow, the ability to scale to meet the rapid pace of new content, and it was costly to adapt to the requirements of working with different institutions, archives, and libraries. Image processing was based upon widely available open-source libraries, which provided fast integration, but often poor performance and poorly documented limitations.

Beginning in 2006, Ancestry.com recognized the need to develop a system which would allow for an adjustable and fluid workflow, highly scalable processing capabilities, and the ability to adapt to unique project requirements.

Solving the workflow problem

When initial requirements and plans were being drawn up for this new system the question was raised as to how to handle the multi-step workflow most digitization requires. DPS operations management and software development teams collaborated to devise a solution.

Traditional projects had been handled by implicit workflows controlled primarily by people and processes, and not by any automated systems. This was not a tenable solution for processing large and diverse quantities of documents quickly. Research led to a decision to use a third-party commercial workflow and toolset, and adapt it where possible.

This solution was a significant advance as it imposed structure and an established framework with which new processes and software tools could be defined. Over a few years, however, it quickly became apparent that this commercial product was simply not designed with the ability to scale beyond a moderately sized operation. The perceived benefits of working with a third-party vendor and the associated support agreements were not as great as anticipated. DPS found work slowed, and sometimes halted, while it waited on a third party to discover and patch bugs. In addition, new and increasingly more complex image processing and workflow requirements were made more difficult or impossible due to limitations in the third-party software suite.

Using this third party product was a good way to launch a new system, but DPS learned quickly that it needed to prepare to outgrow products which could not scale with it. After further research into existing commercial and open-source projects, Ancestry.com's DPS group began moving forward on the development of a new, proprietary, workflow system adapted specifically with scalability, adaptability, and stability in mind.

Joint Work Flow (JWF)

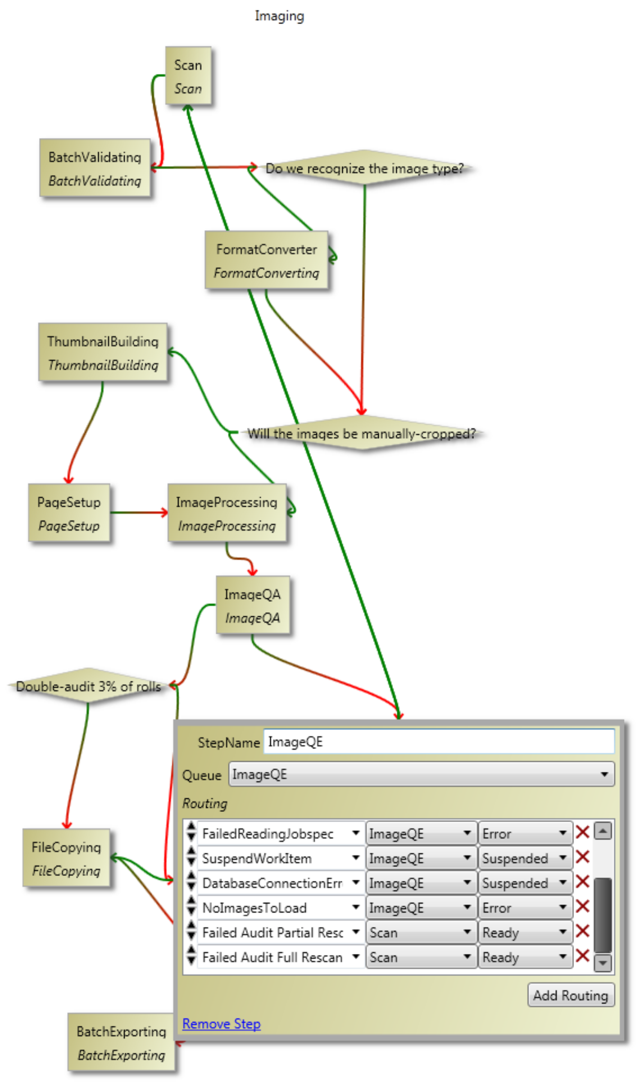
The philosophy behind JWF was that due to the incredibly diverse nature of content, combined with equally diverse requirements, it was impossible to anticipate every future workflow path. As such, the workflow system must be designed to allow extreme flexibility. As the size of operations grew, the necessity of creating a workflow system which operations teams could own and configure without engineer involvement became rapidly apparent.

Under JWF, a workflow is created using a GUI based authoring tool and is a combination of nodes with outcome codes. An example workflow can be found in Figure 1. A node can be as simple as routing a work item to the next queue in a linear flow upon a condition of ‘success’ or to the previous queue upon ‘failure’. It can also allow for an arbitrary number of conditions with associated metadata. A node can even be as simple as a decision point. The advantage of this is that it allows non-technical operational personnel to create and maintain new workflow paths, but also allows for incredibly fast changes to workflow paths, even while a project is in mid-stream.

The way this is implemented dictates that all automatic and manual tools are associated with a workflow queue, which has a corresponding node in the workflow designer. If the requirement were that a collection of books be scanned, prepared for a quality audit, then audited, and finally exported in multiple different formats, the workflow can be set up in a linear fashion in a few minutes. The workflow would then be initiated by a manual tool, in this case ‘Scan Manager’, which would call into the SQL Server based JWF, create a new work item, log statistics, and close the work item as complete. JWF would then automatically move this work item to the next queue. After this, automated software would pull the image location, process the images, and prepare them for a manual quality audit. A failure in the quality audit tool would return a failure code back to JWF, which would then automatically re-route the work item back to the initial scan queue for a re-scan.

Another major feature of JWF is that all routing logic is embedded directly in the workflow, and not within individual pieces of software. For example, DPS has a policy of performing a quality audit on 100% of images scanned, and a double-blind audit on 3% of passed work items. Since this logic is embedded in the workflow, a production supervisor can easily adjust this double-blind percentage to a higher number without delaying our breaking in-process work.

There are several benefits of a system like JWF. The first benefit is that the performance and implementation meet Ancestry.com’s specific needs. If complex new routing capabilities are required, there are in-house software engineers who have written the system, and modifications can quickly be made. There is no waiting on third-parties to respond to support requests or hoping that new, future, features will solve current production requirements. Removing this external reliance allowed Ancestry.com to digitize content far more rapidly, with greater dependability, and with a significantly easier to use system than previously. Second, while all logic and policy are centralized, interaction with the workflow is based on a pull model. The benefit of this is that a central location exists for control, but most work is distributed, allowing for high scalability. Finally, while it was not immediately obvious to DPS, easy workflow and policy management has been essential in allowing production to scale.



StepName ImageQE

Queue ImageQE

Routing

FailedReadingJobspec	ImageQE	Error
SuspendWorkItem	ImageQE	Suspended
DatabaseConnectionErr	ImageQE	Suspended
NoImagesToLoad	ImageQE	Error
Failed Audit Partial Resc	Scan	Ready
Failed Audit Full Rescan	Scan	Ready

Figure 1*. Example of GUI driven workflow from JWF’s workflow designer with the ‘ImageQE’ node selected.*

Setup and Configuration

While JWF determines how work is routed it does not specify how work is completed, by design. One of the unexpectedly high costs in setting up an image processing system is the amount of work that goes into defining what should happen to each collection.

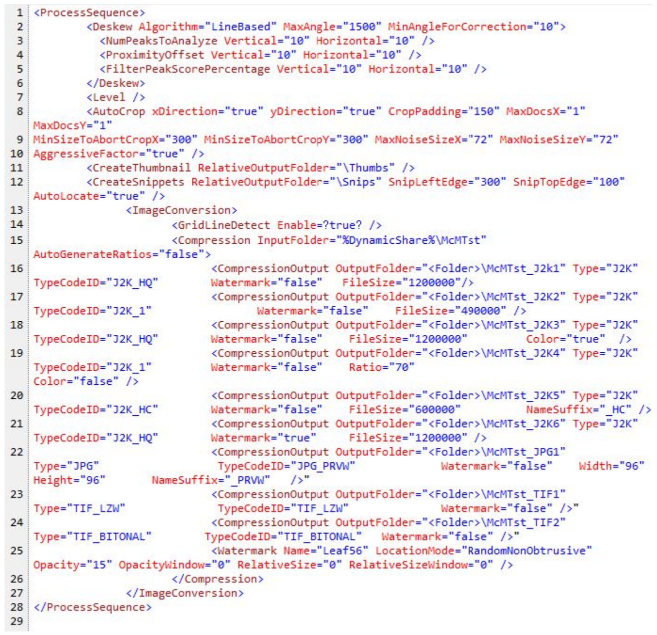
DPS has designed a system which uses XML and inheritance to describe how projects, batches in a project, and images should be worked on in each step. When a project is first defined an overall project template is created which specifies all settings for the specific project. If desired, this specification can be saved as a template, significantly reducing future setup times. As each roll of microfilm, book, or other piece of media is scanned and enters the system, an individual specification is derived for that item from the parent project specification. The benefit of this is that top-level control is maintained, but settings can be overridden as necessary.

The thousands of collections that DPS has processed have reaffirmed the necessity of adaptability in setup and processing. For example, Ancestry.com delivers lossless high-quality images to many of its partners. However, different partners have different delivery requirements. Using templates combined with a very simple setup process allows for rapid adaptation to specific needs of external partners, while still delivering images with very specific settings to Ancestry.com's website.

Image Processing

Historically, image processing has been accomplished via scripts which have called fully open-source software libraries. This system did not scale very well, did not offer much adaptability, and was notorious for poor operational reliability.

After analysis of trends in computing, in 2006 DPS's software development teams decided to create a system which became known as 'Job Server'. The concept behind Job Server was to create a database driven central controller for image processing 'jobs'. Different 'Processors' would integrate with workflow, describe image processing jobs per image in an XML format, and then pass these jobs to Job Server for disbursement. An example of the container for a job submitted to iFarm can be found in Figure 2. A large number of image processing 'agents' would run on windows servers, and would periodically call into Job Server asking for work. Every agent was designed to fulfill any and all image processing tasks.



```
<ProcessSequence>
        <Deskew Algorithm="LineBased" MaxAngle="1500" MinAngleForCorrection="10">
          <NumPeaksToAnalyze Vertical="10" Horizontal="10" />
          <ProximityOffset Vertical="10" Horizontal="10" />
          <FilterPeakScorePercentage Vertical="10" Horizontal="10" />
        </Deskew>
        <Level />
        <AutoCrop xDirection="true" yDirection="true" CropPadding="150" MaxDocsX="1"
MaxDocsY="1"
MinSizeToAbortCropX="300" MinSizeToAbortCropY="300" MaxNoiseSizeX="72" MaxNoiseSizeY="72"
AggressiveFactor="true" />
        <CreateThumbnail RelativeOutputFolder="\Thumbs" />
        <CreateSnippets RelativeOutputFolder="\Snips" SnipLeftEdge="300" SnipTopEdge="100"
AutoLocate="true" />
              <ImageConversion>
                     <GridLineDetect Enable=?true? />
                     <Compression InputFolder="%DynamicShare%\McMTst"
AutoGenerateRatios="false">
                            <CompressionOutput OutputFolder="<Folder>\McMTst_J2k1" Type="J2K"
TypeCodeID="J2K_HQ"         Watermark="false"   FileSize="1200000"/>
                            <CompressionOutput OutputFolder="<Folder>\McMTst_J2K2" Type="J2K"
TypeCodeID="J2K_1"                 Watermark="false"    FileSize="490000" />
                            <CompressionOutput OutputFolder="<Folder>\McMTst_J2K3" Type="J2K"
TypeCodeID="J2K_HQ"         Watermark="false"    FileSize="1200000"         Color="true"  />
                            <CompressionOutput OutputFolder="<Folder>\McMTst_J2K4" Type="J2K"
TypeCodeID="J2K_1"          Watermark="false"    Ratio="70"
Color="false" />
                            <CompressionOutput OutputFolder="<Folder>\McMTst_J2K5" Type="J2K"
TypeCodeID="J2K_HC"         Watermark="false"    FileSize="600000"          NameSuffix="_HC" />
                            <CompressionOutput OutputFolder="<Folder>\McMTst_J2K6" Type="J2K"
TypeCodeID="J2K_HQ"         Watermark="true"     FileSize="1200000" />
                            <CompressionOutput OutputFolder="<Folder>\McMTst_JPG1"
Type="JPG"                   TypeCodeID="JPG_PRVW"            Watermark="false"    Width="96"
Height="96"       NameSuffix="_PRVW"   />"
                            <CompressionOutput OutputFolder="<Folder>\McMTst_TIF1"
Type="TIF_LZW"               TypeCodeID="TIF_LZW"             Watermark="false" />"
                            <CompressionOutput OutputFolder="<Folder>\McMTst_TIF2"
Type="TIF_BITONAL"         TypeCodeID="TIF_BITONAL"   Watermark="false" />"
                            <Watermark Name="Leaf56" LocationMode="RandomNonObtrusive"
Opacity="15" OpacityWindow="0" RelativeSize="0" RelativeSizeWindow="0" />
                     </Compression>
              </ImageConversion>
</ProcessSequence>
```

***Figure 2**. An example of a parent task XML describing everything that should happen with a source image.*

Job Server allowed DPS to begin to process images at an astounding rate and with scalability it had not seen before. However, over time it became apparent that Job Server had some limitations which prevented full utilization of image processing hardware. The major limitation was that while a collection of a thousand images may be divided into 200 5-image jobs, no new processing of another batch in that step could commence until the current job finished. This had the effect of image processing agents sitting idle while there was plenty of work to be done.

iFarm

To fix this scalability issue, a rewrite of the system commenced in 2008, named 'iFarm' for Imaging Farm. iFarm was significantly different and more advanced for several reasons. First, while the processors still submitted jobs to iFarm in the same way, they were written to be multi-threaded. They would simultaneously pull new work items, describe image processing in XML, submit this to iFarm, and wait for status. The practical outcome of this was that processors were constantly creating and clearing jobs with little to no down time. Second, combined with this, iFarm no longer maintained a one-to-one relationship with processors. iFarm would accept and work on jobs to a database defined job limit. Third, the agent was rewritten to be even more agnostic about how it completed work allowing for additional work, such as pre-flight and post-flight processes to be easily distributed.

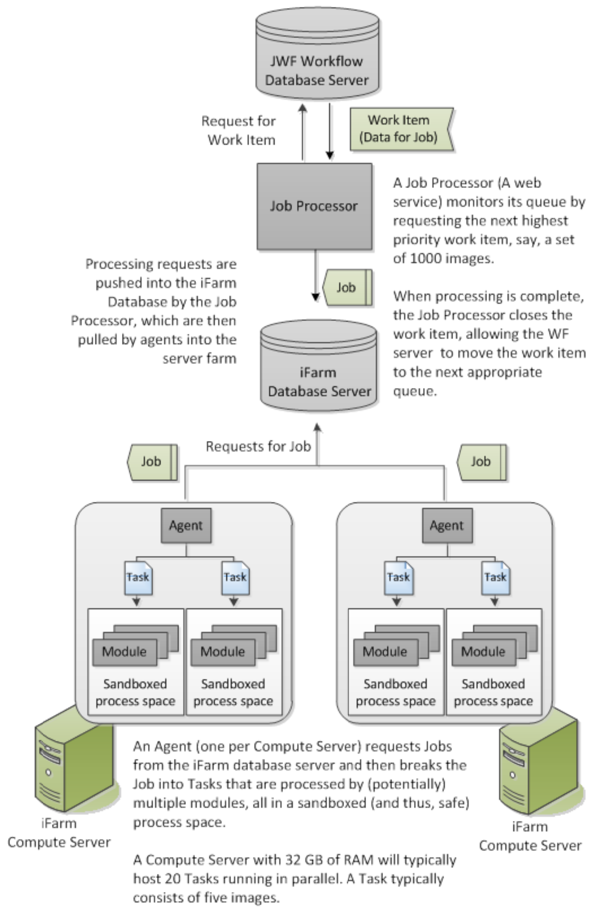

JWF Workflow Database Server
Request for Work Item
Work Item (Data for Job)
Job Processor
A Job Processor (A web service) monitors its queue by requesting the next highest priority work item, say, a set of 1000 images.
Processing requests are pushed into the iFarm Database by the Job Processor, which are then pulled by agents into the server farm
Job
When processing is complete, the Job Processor closes the work item, allowing the WF server to move the work item to the next appropriate queue.
iFarm Database Server
Requests for Job
Job
Job
Agent
Task
Task
Module
Module
Sandboxed process space
Sandboxed process space
Agent
Task
Task
Module
Module
Sandboxed process space
Sandboxed process space
An Agent (one per Compute Server) requests Jobs from the iFarm database server and then breaks the Job into Tasks that are processed by (potentially) multiple modules, all in a sandboxed (and thus, safe) process space.
A Compute Server with 32 GB of RAM will typically host 20 Tasks running in parallel. A Task typically consists of five images.
iFarm Compute Server
iFarm Compute Server

***Figure 3**. Diagram depicting the flow of job to the iFarm*

Initially, a change to image processing code required the farm to be brought down, new code rolled, and brought back up. The cost of stopping all production made it expensive to roll new code or features. To solve this problem, the agent was rewritten so that a command could be passed to pull the latest versions of all image processing modules upon the completion of its current work, reload itself, and resume working.

The initial roll from Job Server to iFarm resulted in an immediate 600% increase in automatic production capacity with no increase in image processing hardware. In addition, production downtime was eliminated and replaced with a minimal, temporary, rolling reduction in capacity. This new design not only allows for extremely rapid image processing speeds, but makes it relatively inexpensive to add new capabilities to the image processing pipeline. Code can be written as an individual kernel, and rapidly rolled into the iFarm architecture allowing for immediate scalability for even very complex tasks. As time has progressed, iFarm has also been generalized to allow processing of any type and is no longer image-centric allowing for another massive breakthrough in aggregate production rates.

Build, Buy, or License?

One of the most important questions involved in software development of this type is when to build, buy, or license. At the lowest level within DPS's image pipeline is the image processing 'agent'. The agent is built from a combination of open-source, proprietary, and licensed third-party software.

Initially, primarily open-source software was used due to minimal cost of implementation and broad, community-driven feature sets. As DPS software development engineers compiled image processing requirements, they broke down items into core technologies and ancillary technologies. For example, there is very little benefit in reinventing the wheel where decoding images is concerned. There already exist plentiful libraries, open-source and commercial, for decoding all types of images and DPS leverages open-source technologies with no modification. In addition, simple tasks like auto-leveling, rotation, and cropping exist in similarly plentiful libraries. However, when working with certain types of documents, product requirements call for as near to perfect de-skew, or alignment, on the images as possible. Unfortunately no libraries, commercial or otherwise, were available which met Ancestry.com's specific needs. Time was invested in researching and solving the problem, and a unique and proprietary module was created to detect the precise angle of rotation needed to meet the very specific requirements. A further example is that when images are exported to Ancestry.com, the website has very specific requirement. The investment in developing the technology in-house would have been cost prohibitive. Fortunately a commercial solution was found to the problem which had an added benefit of being 40% faster at encoding images than our previous solution, also a commercial product. In this case, the library was licensed so that other, more domain specific, work could continue.

There are two very important lessons that were learned over the course of 8 years of developing Ancestry.com's current image processing systems. First, speed of delivery usually trumps efficiency of code. While three months could be spent optimizing a particular piece of code in a lower level language like C++, similar, albeit less performant, code could be written in a language like C# and additional computing power can be used to make up the difference. This means that while individual pieces of code may be less optimal, the aggregate product advances, and thus the operational speed and quality improve far more quickly. When absolutely necessary, specific sub-modules can be optimized where heavy lifting efficiency is needed. Second, resources should be allocated to where the core competencies and requirements of the business are, and not on unnecessary side-projects. A lot of time can be spent developing highly optimized systems which do very little to move a project forward.

Storage Solutions

There are many hardware solutions which allow for fast access or large bandwidth, but not many which cross that divide. DPS has experimented with many different hardware solutions for storage and processing over the past 8 years.

Initially resources were put into combinations of different types of Network Attached Storage (NAS) and Storage Area Networks (SANs). Implementations of different brands of hardware were problematic. Frequently, the largest bottlenecks in DPS's image processing operations were disk I/O on NASs or lack of sufficient work in process (WIP) storage space. As image processing capacity scaled into hundreds of simultaneous image processing threads, the storage could not keep up and was not easy to scale. The ultimate issue was that traditional storage was designed to 'scale-up', not to 'scale-out'.

A commonly solution offered was to move image processing into an externally hosted cloud. However, this was simply not feasible for many reasons. Primarily, the costs of bandwidth and storage for the processing of so many high-quality, losslessly compressed images would be in the hundreds of thousands to millions of dollars per year. For high CPU, but low bandwidth applications, cloud solutions have much to offer. However, image processing is an extremely high bandwidth, high storage, and an extremely CPU intensive process making it cost prohibitive for external cloud solutions.

Another solution frequently suggested was to use technologies like Hadoop or other map-reduce schemes to distribute processing and storage. These technologies are fantastic for data analysis and condensation, but it was discovered quickly they were poorly suited DPS's image processing and analysis needs. With a task like image processing, the combination of incredibly intensive CPU, memory, storage, and I/O made schemes like this very difficult to scale.

DPS began experimenting with a new clustered storage provider. The advantage of using a clustered storage solution comes from the ability to scale storage, speed, and bandwidth as needed in a single addressable system.

Initial testing began in 2008, as an evaluation cluster was deployed. Benchmarking was extremely encouraging, and within a few months the evaluation cluster had become a mainstay of production infrastructure. Most storage in DPS is now clustered storage.

Currently DPS WIP storage totals nearly a petabyte split into four separate Isilon clusters. Under a moderate load, these systems are able to maintain an average of more than four gigabytes per second of throughput, serving hundreds of image processing machines and dozens of production employees in real-time. One of the greatest benefits of using clustered storage is that specific nodes within a cluster can be crafted to meet needs. For example, DPS's primary image processing cluster is hit with heavy I/O as well as frequent directory reads. We discovered during profiling of our processing systems that one of our bottlenecks was waiting for directory reads to return so that jobs could be built. So, a decision was made to use Isilon IQ X-Series nodes which use solid state drives (SSDs) for file system metadata, and hard drives (HDDs) for the bulk of storage. Utilizing this hardware immediately removed a bottleneck from our image processing systems with no changes to image processing or workflow code.

Computing Solutions

For image process computing, DPS initially used inexpensive generic hardware. These machines were former web servers running Windows Server 2003. One of the counter-intuitive outcomes of beginning with hardware like this was that it was so unreliable that image processing systems had to be written with incredible robustness and no single machine could be allowed to be a break-point. What resulted was a system which was robust in-spite of hardware. DPS has discovered over time that this level of robustness pays huge dividends even with highly reliable hardware and infrastructure.

As production requirements grew, a significant amount of experimentation was put into finding the right type of hardware to use going forward. Conventional wisdom would dictate that the correct solution would be to continue to use inexpensive hardware, as opposed to highly specialized and highly expensive processing hardware. Tests showed that the most economical approach to processing was actually in the middle of the two. The management, power, and maintenance costs of using extremely inexpensive hardware were not upfront, but they were very real. However, with more specialized hardware, you could not get the same performance per dollar and had to buy in less granular increments. In the end, it was discovered that the most efficient usage of resources was to use mid-tier servers. Currently, we use servers which utilize dual Intel Xeon processors and copious amounts of ram. The practical effect of this is that 20 image processing threads can be run simultaneously per machine without taking system resource utilization above 93% on a full-load.

The most important lessons which were learned were that it was highly important to be data-driven when making decisions about how to best allocate resources. Conventional wisdom and wider computing industry best practices were not always applicable to our specific needs.

Conclusion

Ancestry.com has faced challenges which are often unique to working in the document preservation field but are highly applicable to other institutions in the same space. DPS has found success through focusing on creating systems which meet its unique needs, but avoiding unnecessary duplication of work.

There are three main themes, or cultural ideals, which have allowed DPS to be successful. First, it has been highly important to be open-minded about possibilities and solutions. It is tempting to commit to established systems and concepts or to avoid the expense of change by living with a sub-optimal system. However, the result can often be stagnation. Second, creating a culture where trial and error are key components in system development has allowed for innovations and dramatic improvements which might not have been found otherwise. Third, being data-driven in decision making is paramount. Many counter-intuitive, but nonetheless correct, choices have been made because of data and benchmarking which have contradicted prevailing opinion.

Solving the problems of making operations scale and ensuring they thrive is not always intuitive and is rarely easy; problems easily solved do not often provide dramatic results. Tackling these difficult problems with the seriousness they deserve will allow an institution to make dramatic leaps toward its goals.

Author Biography

Joshua B Harman is the Systems Manager for Ancestry.com's Digital Preservation Services organization and is responsible for managing the workflow technology, systems, and software which digitize, process, and index data for Ancestry.com and other partners.