To the Cloud and Back: A Distributed Photo Processing Pipeline

Nicholas Jaeger and Peter Bui
Department of Computer Science
University of Wisconsin - Eau Claire
Eau Claire, WI 54702
{jaegernh, buipj}@uwec.edu

Abstract

In this project, we constructed a distributed photo processing pipeline that utilized Dropbox to gather incoming photos, Python-WorkQueue to distribute image processing tasks to a local Condor cluster, and a fault-tolerant daemon that manages these components to automatically process and archive the large collection of data. This system was made to assist a team of archaeologists working in Israel, where collaboration is more difficult, and internet access is very limited. Our paper details the design and implementation of the system and evaluates its effectiveness in the field test.

1 Introduction

In this paper, we present our distributed photo processing pipeline (DP3). This sysem is the end result of needing an effective way to collaborate and archive photos for a project in Israel. The project was an archaeological dig that took place in December of 2012, involving professors from around the world and working at different locations. Some sites had Internet access, while other places did not. For the locations that did have Internet access, the upload speeds were often quite low (as low as 0.5 Mbps). Moreover, some of the members of the archaelogical project did not use computers often and desired an easy to use system for collaborating and archiving the photos they took at the dig sites. Furthermore, the system needed to be automated to take advantage of Internet connections when they became available (without requiring the user to start a transfer manually).

To address the needs of the archaeological dig, this project developed a hybrid distributed system consisting of both cloud storage and traditional local clusters to process photos for the project in Israel. Popular standalone options for tasks like this include Picasa [2] with online syncing and the Dropbox [1] photo application. Unfortunately, both of these options have storage limits unless users pay premium fees for increased resources. For this project, it was important to come up with a solution that could handle upwards of 50gb of photos, while relying on storage space already provided by a university to minimize project costs and avoid paying premiums.

To address this challenge, we developed DP3, an elastic and fault-tolerant distributed photo processing pipeline that utilizes Dropbox for initial storage, Python-WorkQueue [6] for coordinating distribute computation, and Condor [8] for managing local system resources. This system collected the photos in a central archive, processed them, and provided access to them via a web portal. In this paper, we examine the usability and shortcomings of our system.

2 Design

An overview of the architecture of our distributed photo processing pipeline is provided in Figure 1.

1. **Transfer:** The first phase of the system involves users transferring their photos to personal laptops.

2. **Upload:** Next, the files are uploaded to Dropbox. Each user has a folder named after them in the dropbox respository used. This enables the system to tag the pictures with the photographer's name.

3. **Monitor:** A daemon running on a server at the University of Wisconsin - Eau Claire monitors the Dropbox repository for new files. When incoming photos are detected, the daemon dispatches the photo processing tasks to the local distributed computing cluster.

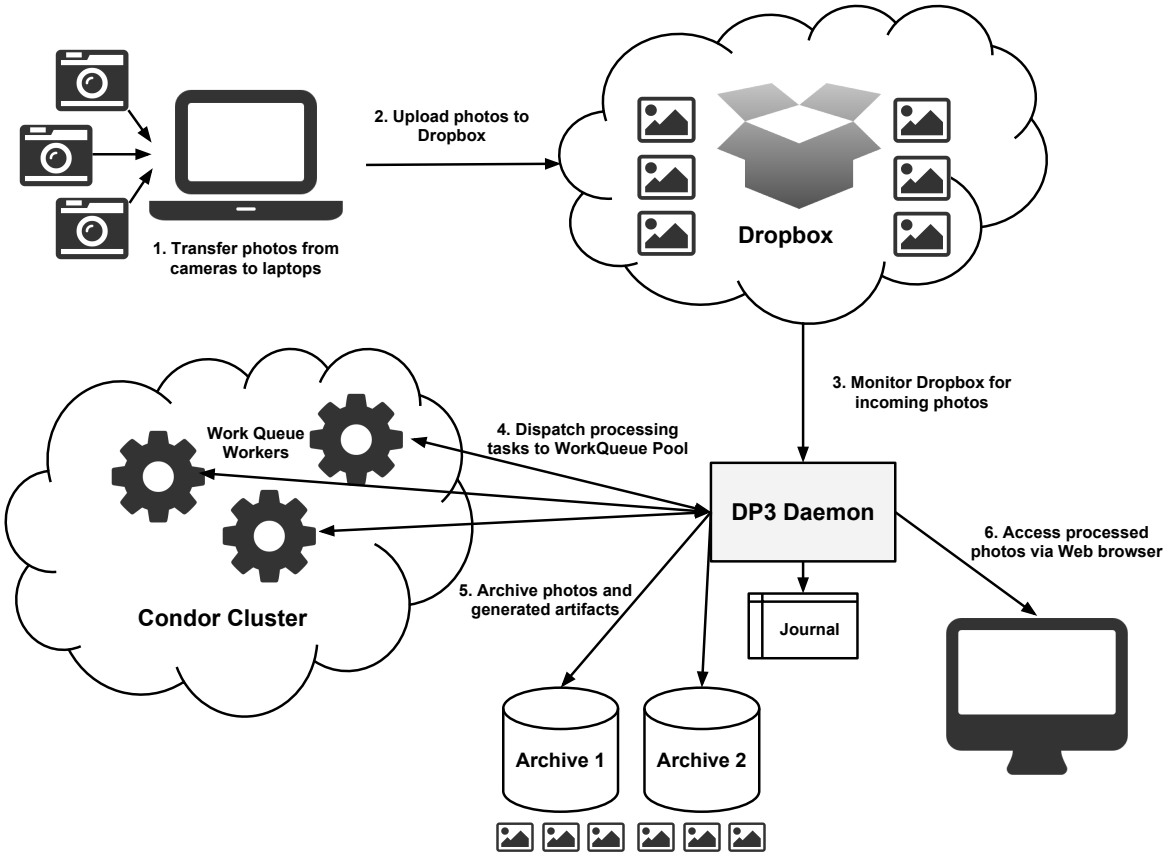

Figure 1: Distributed Photo Processing Pipeline

4. **Process:** Tasks are scheduled and dispatched to remote workers running on the local Condor cluster. These processing tasks include making thumbnail and web-optimized (640x480px) versions of the pictures.

5. **Archive:** When these tasks are completed the original photos and the generated artifacts are archived to multiple storage systems for redundancy.

6. **Collect:** Once the images and artifacts are archived, the originals in Dropbox are removed, freeing space for more images. This enables us to use a free dropbox account with limited storage space.

7. **Publish:** The daemon also serves as a web portal that allowed users to track and access the source images and the generated outputs. When a job is finished a web-page is generated using thumbnails and metadata from the source images to give researchers an organized and visual display of all the project photos.

Once the DP3 monitor daemon and Dropbox client are started on a machine in the University of Wisconsin - Eau Claire Condor cluster, the system is completely automated. Users simply upload files to the appropriate Dropbox folder and the system will detect the data, process it and archive it.

3 Methodology

To implement DP3, we started by provisioning one of the machines in the local University of Wisconsin - Eau Claire Condor cluster. With the help of the system administrator, we installed the Dropbox client and had it synchronize with the appropriate folder. We then established particular naming conventions on where and how to store upload photos in the shared Dropbox folder.

3.1 Monitor

Next, we wrote a Python daemon that monitored the mounted Dropbox folder for new photos to process. To do this, it periodically walked the Dropbox folder and checked for any files. Because photos would be collected or removed from the Dropbox folder after they were archived, we simply had to check if the file was an image file (i.e. had a `.jpg` extension) and was not one of the artifacts we generated as part of the pipeline (i.e. did not contain the strings "WEB" or "THUMBNAIL"). If the file met these criteria, it was scheduled to be processed on the local distributed computing cluster

3.2 Pipeline

Once the DP3 daemon detected a new file, it would progress through the distributed processing pipeline, which consisted of the following stages:

1. **Submit**: Once the file is detected, a processing task is scheduled to run on the local University of Wisconsin - Eau Claire Condor [8].

2. **Process**: When a worker on the cluster becomes available a task is sent to it, and the file is processed to generate a thumbnail sized image of the original photo (120x90px) and a web-optimized version of the photo (640x480px).

3. **Archive**: Once the processing task is complete, the original photo and the generated artifacts are archived to the user-specified locations.

4. **Collect**: If the archival process is successful, then the original photos are removed from Dropbox to free up space for incoming photos.

Each state depends on the previous one, so if the processing failed, then the photo would not progress to the "Archive" state. Likewise, if a photo was processed but not successfully archived, then it would not be collected or removed from Dropbox. This careful use of states was to ensure that only remove files from Dropbox when we were absolutely certain that the photos had been processed and archived.

3.3 Journal

To ensure consistency, the DP3 daemon used a transaction journal to keep track of the state of the system. Whenever a file appeared, the daemon would record its presence in the journal with a JSON entry that looked like this:

```
{"status": "Processed",
 "datetime": 1355517208,
 "archives": ["/data/scratch/dp3/Archive/2012-12-14"],
 "owner": "JAEGER-NICHOLAS",
 "path": "/Dropbox/Photos/Israel/JAEGER-NICHOLAS/DSC00372.JPG",
 "id": "JAEGER-NICHOLAS_1355517208_DSC00372.JPG"}
```

As seen, each journal entry was simply a set of key-value pairs containing the following information:

1. **status**: This stored the current state of the file. As it progressed through the various stages of the pipeline, this field would be updated and recorded in a new journal entry.

2. **datetime**: This stored the timestamp of the file (that is, it's creation time). This information would be used later to organize the files into time-based folders.

3. **archives**: This recorded where the file was archived or stored. Our system allowed for any number of archival destinations, but we only used two when in production.

4. **owner**: This indicated the user who uploaded the file. To enable straightforward identification of the user, we setup a convention where users would always upload to a Dropbox sub-folder named `FIRST_NAME-LAST_NAME`.

5. **path**: This was the original location of the file on the mounted Dropbox filesystem.

6. **id**: This field was generated based on the owner, datetime, and basename of the file and was used as the key to the internal dictionary used by the DP3 daemon to cache the contents of the journal in memory.

Whenever a file transitions from one stage of the pipeline to the next, it is recorded in the transaction journal. This is vital for maintaining consistency and determining what to do when say the daemon crashes. Since it has a journal that only contains entries that are added after they have been completed, then it knows which tasks it can avoid. Therefore if the daemon crashes in the middle of processing a set of files, it knows that once it has recovered, it can skip the already processed files. Moreover if the user accidently re-uploads a previously archived file, then it knows to skip it since it is already processed.

3.4 WorkQueue

To coordinate and organize the processing tasks on the Condor cluster, we utilized the Python-WorkQueue [6] master-worker distributed computing framework. Using this library simplified our photo processing pipeline because WorkQueue manages task scheduling, data transfers, and fault-tolerance with respect to worker failure. This means that we only need to specify the processing executables and data files to WorkQueue and the framework would manage the details of coordinating the execution of tasks and transfering

of input and output data. With WorkQueue it is possible add and remove workers during execution, which means we can scale up or down depending on our processing requirements. This allows us to increase the overall throughput of our workflow by distributing work across multiple machines.

Along with the ability to dynamically add and remove workers, WorkQueue also transparently handles situations where workers may fail or disconnect by rescheduling tasks to run on other workers. Because of this, we did not need to manually handle fault-tolerance with respect to worker failure within our application. Note, a completed task is not the same as a successfully processed task; WorkQueue may return a task that has executed but did not generate the appriopriate artifacts for whatever reason (say it ran out of space on the remote node). To WorkQueue, the task is complete because it executed and returned, but to DP3 this is a failed processing task. This is another reason why the transaction is useful: it allows us to carefully monitor the status of the pipeline.

3.5 Web Portal

Once photos have been processed on the distributed system, archived to various storage locations, and removed from Dropbox, the original photo along with the generated artifacts are made available via a web portal. This was an important component of the system as it provided users easy access to their data and the generated artifacts, and it served as a way of monitoring the overall system. Figure 2 provides an example of the image gallery provided by the web portal.

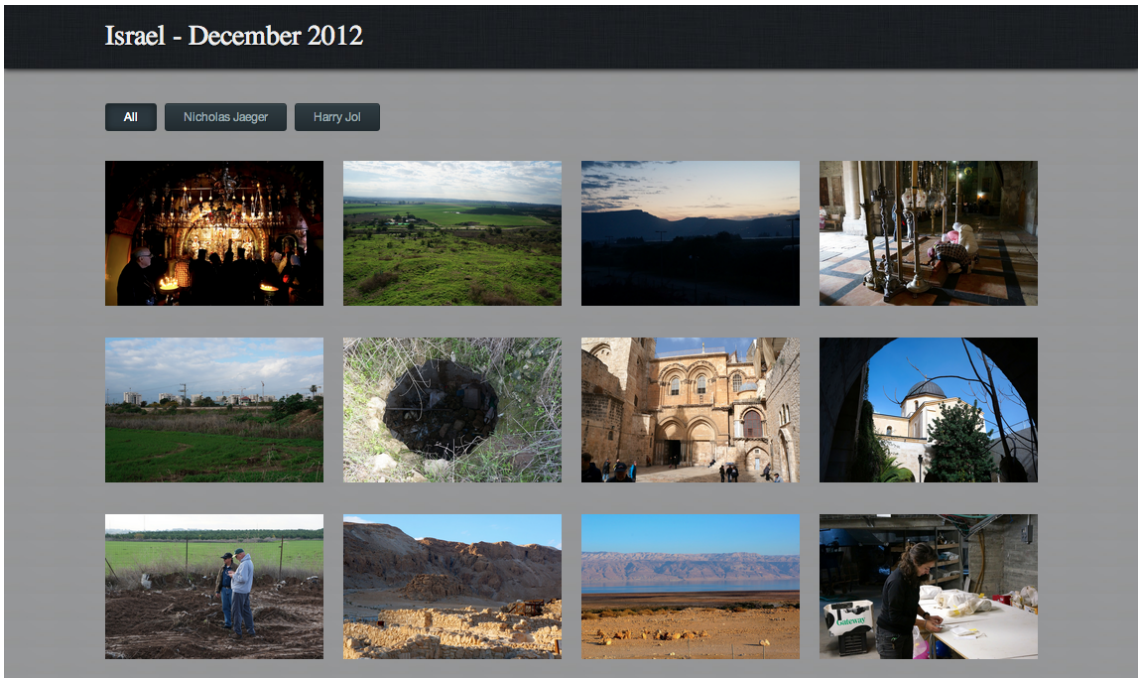

Figure 2: DP3 Web Portal

4 Evaluation

We evaluated the distributed photo processing pipeline based on its utilization in a live field test in Israel. Working with collaborators in geography and archaeology, we configured the system to process images from archaeological excursions in Israel. This included setting up the system to detect file and folder names, as well as metadata and to process those photos. In this experiment, researchers at archaeological sites in Israel transferred their photos to Dropbox at different times to folders named after the photographers. As explained previously, our DP3 monitor daemon detected these files and processed them by generating a set of thumbnails and web-optimized images. These artifacts, along with the source images, were then archived to the researchers' local storage accounts and made access via a password-protected website as shown in Figure 2.

Number of Batches	448
Maximum Size	1385
Minimum Size	1
Average Size	16.4

Table 1: Batch Statistics.

Table 1 provides a summary of the batches of photo processing tasks our system performed. In total, DP3 executed 448 batches or groups of task. The largest such group consisted of 1385 photos, while the smallest consisted of just 1. The average number of tasks in each batch was 16. The reason for these batches was that our monitor only checked the Dropbox directory every 5 minutes in order to collect as many photos as possible and prevent us from polling the Dropbox filesystem too often.

Number of Tasks Submitted	7372
Number of Tasks Failed	104

Table 2: Task Statistics.

Table 2 shows a summary of the number of tasks executed as part of the distributed photo processing pipeline. In total 7372 task were submitted. Of those, 104 resulted in failures (about 1.4%). Some of these failures were due to bad file naming on the part of the users (e.g. 'REEDER-PHILIP_1355825903_(2)2009.YAVNE.AREA.PROPOSED.INSERT.JPG'). Other failures were because we hit machines without the necessary libraries for our processing tasks. Because of our transaction journal we able to either ignore these errors as in the first case, or temporarily suspend the application, fix the problem, and restart it without duplicating work or losing data.

Using a pool of 2-16 workers which we managed manually (i.e. we added or removed workers based on our observations of the system load), the system processed 5397 files over the course of two months. This equated to 31GB of data archived in two storage systems: one in the local Condor cluster and a second one on the University of Wisconsin - Eau Claire shared network drive. We were able to use a dropbox account with only 7.9GB of available space by removing original images from the repository after they were processed and archived.

The weakness in the system that ultimately limited performance was the upload speeds available to the system users. Based on Internet speed tests, upload speeds ranged from 0.46 to 3.74 Mbps [3]. Not all of the users understood how to change the Dropbox programs settings, so the upload speeds were further throttled by the Dropbox software, which only took advantage of 75 percent of the available upload speeds [1]. Because of this, the Condor cluster remained largely under-utilized due to the lack of data to process. That said, the overall system design was a success as it provided an automated and easy-to-use distributed photo processing pipeline with robust and fault-tolerant features.

5 Related Work

This system is very comparable to the Dropbox photo application, but utilizes a distributed computing cluster to increase the throughput of the photo processing workflow. Additionally, our system is designed to carefully archive the photos to multiple locations and utilizes a transaction journal to maintain consistency.

In some ways, DP3 is comparable to a system like Biocompute [7], which is a web-based resource that uses grid computing for processing bioinformatics problems. Similar to DP3, Biocompute farms tasks out to a Condor computing grid and presents an easy-to-use web portal for accessing and managing data.

Another comparable management and repository system is BXGrid [5]. BXGrid is a web portal that displays galleries of thumbnails of different types of biometric data. It also utilizes a distributed data processing pipeline in the back-end to generate artifacts. DP3 is different from BXGrid in that it archives data directory to the filesystem rather than through a meta-filesystem such as ROARS [4] and it only focuses on photos rather than a variety of biometric data.

6 Conclusion

The DP3 system satisfied the needs of the archaeological team by carefully archiving photos in a robust manner and providing Internet access to them. The archaeologists also found it very easy on their end, which was important. Rather than having users worry about trying to resize and add their photos to the archaeological dig's collection, the researchers simply drag-and-dropped their photos into the correct Dropbox directory, and DP3 handled the en-

tire photo processing pipeline. The main obstacle proved to be the Internet upload speeds in Israel were too slow and thus could not take full advantage of the system. Unfortnately, there was no way around that for the archaeologists utilizing the system. For archaeological work at sites with better internet speeds, this system would not be limited in this way, and would be noticeably more effective.

Overall, the distributed photo processing pipeline was shown to be an effective approach of allowing users to take advantage of both cloud computing resources and local distributed cluster systems. By combining the two, we developed an effective and useful system for collaborating and processing large amounts of digital images in an automated and easy-to-use manner.

References

[1] Dropbox. http://www.dropbox.com/, 2013.

[2] Picasa. http://picasa.google.com/, 2013.

[3] SpeedTest.Net. http://speedtest.net/, 2013.

[4] H. Bui, P. Bui, P. Flynn, and D. Thain. ROARS: A Scalable Repository for Data Intensive Scientific Computing. In *The Third International Workshop on Data Intensive Distributed Computing at ACM HPDC 2010*, 2010.

[5] H. Bui, M. Kelly, C. Lyon, M. Pasquier, D. Thomas, P. Flynn, and D. Thain. Experience with BXGrid: A Data Repository and Computing Grid for Biometrics Research. *Journal of Cluster Computing*, 12(4):373, 2009.

[6] P. Bui, D. Rajan, B. Abdul-Wahid, J. Izaguirre, and D. Thain. Work Queue + Python: A Framework For Scalable Scientific Ensemble Applications. In *Workshop on Python for High Performance and Scientific Computing at SC11*, 2011.

[7] R. Carmichael, P. Braga-Henebry, D. Thain, and S. Emrich. Biocompute 2.0: An Improved Collaborative Workspace for Data Intensive Bio-Science. *Concurrency and Computation: Practice and Experience*, 23(17):2305–2314, 2011.

[8] D. Thain, T. Tannenbaum, and M. Livny. Condor and the Grid. In F. Berman, A. Hey, and G. Fox, editors, *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley, 2003.