Storage and Development of Digital Image Negatives in a Distributed File System

Matthew Malensek and Shrideep Pallickara
Department of Computer Science
Colorado State University
Fort Collins, USA
{malensek, shrideep}@cs.colostate.edu

***Abstract*—Steady innovations in traditional computing hardware have had a profound effect on other consumer electronic devices in use today; as processing becomes cheaper and storage capacities become larger, users are producing data at an unprecedented rate. To cope with these new storage demands, distributed and cloud-based systems are increasingly popular due to their scalable, parallel nature. In this paper, we investigate the use of cloud computing technology to provide storage and processing capabilities for extremely large image-based datasets and photographic editing workflows, and demonstrate the performance gains provided by such a system.**

***Index Terms*—Distributed file systems, commodity clusters, image processing, scale-out architectures**

I. Introduction

The steady evolution of digital photographic equipment has led to increased storage and processing demands on modern computing hardware. These demands largely surpass the capabilities of commodity desktop PCs, resulting in more photographers seeking distributed storage and processing solutions for their digital images. State-of-the-art digital single-lens reflex (SLR) camera sensors can produce photographs upwards of 36 megapixels, and a fast-paced sports shoot could be comprised of hundreds of images. Traditionally storage area networks (SANs) are used to cope with these large datasets, but they require users to transfer images to their workstation before processing can begin and then save the results back to the SAN. In this paper we propose an alternative approach based on commodity clusters, and aim to exploit both their storage *and* processing capabilities to reduce the amount of processing done at individual workstations.

While several image formats provide compression for these collections of large images, there are many advantages to working with raw sensor information, also known as *digital negatives*, directly from the camera. These *raw* files allow for much more flexibility in a post-production environment, where image attributes can be modified to produce subjectively better artistic results. In addition, most compressed image formats are lossy, resulting in decreased fidelity of the stored image. Unfortunately, the process of converting raw sensor data to viewable images, referred to as *developing* the image, can be quite computationally expensive. This is largely due to the complexity of reconstructing an image from sensor data rather than simply reading pixels to be displayed on screen. To further exacerbate this problem, "non-destructive" editing of digital images, where modified versions of a file are not written over the original, requires a linked history of edits to be stored with each image and then applied on-demand rather than being read from disk directly.

A. Research Challenges

For this study, we aim to provide an alternative approach to the expensive, powerful hardware required for modern professional photo editing workflows. Instead, we leverage the collective power of a cluster of commodity hardware for storage and development of digital images. This research goal provides a number of challenges:

1) *Processing*. Developing digital negatives is a computationally-expensive task. Therefore, a divide-and-conquer approach should be used to dispatch work across a number of processing elements in parallel.
2) *Organization*. While it is possible to evenly disperse images across an entire cluster, the images often contain logical groupings, such as shoot dates or locations. These related files should be spread across a small number of computing resources to reduce latency.
3) *Edit history*. A number of modifications and transformations can be applied to raw image data. Therefore, each step in the editing process should be recorded to allow users to add or remove edit steps without modifying the original source image.
4) *Retrieval*. Users should be able to request files from the system that match a number of metadata parameters using both ranges and exact values of attributes, and then apply arbitrary image-processing computations across the results.

B. Paper Contributions and Organization

Our paper illustrates the viability of storing a large dataset composed of digital negatives across a number of computing resources while also applying computations to perform image manipulations on the files in a non-destructive manner. The rest of the paper is organized as follows: Section II describes the components of our system and how they interact. Section III investigates the best balance of storage load in the system, followed by a description of our non-destructive

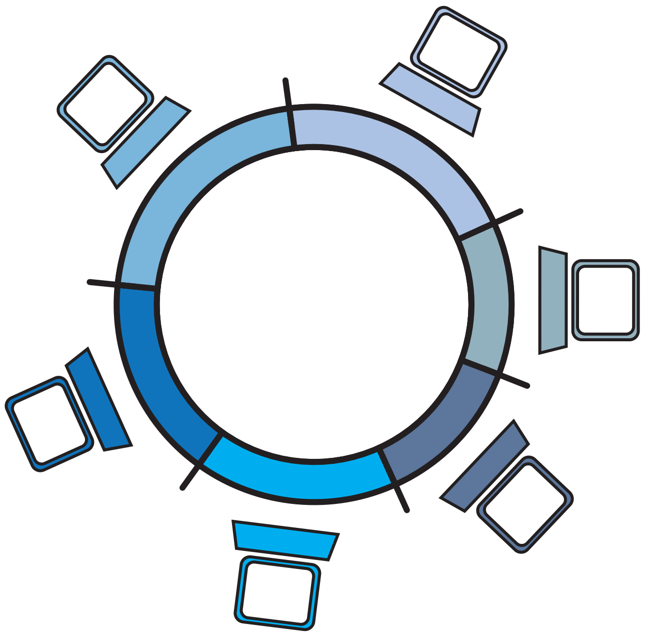

Fig. 1. A simple example Distributed Hash Table network, with a hash space divided among six nodes.

editing capabilities in Section IV. We bring the paper to a close with performance benchmarks in Section V followed by our conclusions and future work in Section VI.

II. System Overview

Our image storage and development framework consists of two primary components: *Galileo*, our distributed storage system, and *Granules*, our distributed stream processing system. These two components distribute storage and processing operations across arbitrarily-sized and geographically-diverse clusters of computing resources.

A. Galileo

While Galileo [1], [2] has primarily been used for storing multidimensional atmospheric data, it can handle extremely large datasets from other fields as well. Galileo provides storage, replication, and indexing functionality for files with high dimensionality, with these dimensions (also known as *features*) stored as metadata in the system. Using the metadata, Galileo also provides fast exact-match and range-based query support for retrieval operations [3].

Galileo is organized as a Distributed Hash Table, (DHT) and provides several enhancements over traditional DHTs. DHTs are decentralized and highly scalable, making them ideal in a distributed setting. A DHT partitions a hash space among a group of computing resources, and then uses an associated hash function to provide storage and retrieval operations similar to a hash table data structure. Figure 1 provides an example of a simple DHT network with a hash space divided among six nodes.

Files in Galileo are called *blocks*, which can contain arbitrary collections of data. Each block has an associated metadata block, which contains additional dimensions and information about the block. Blocks and their metadata are replicated across the cluster to prevent data loss in the case of hardware failure or network partitions.

B. Granules

Granules [4] is an open-source, lightweight distributed stream processing system. Granules can express computations using MapReduce or directed, cyclic graphs for incremental processing operations that build state over time. In this work, we used MapReduce computations to divide tasks over a number of processing elements. These computations are scheduled and run across available computing resources in a cluster and can be directly launched by Galileo through a user-friendly API. Computations can be developed in multiple languages, including C, C++, Java, Python, and R. Granules has been deployed in a number of domains, including brain-computer interfaces [5], handwriting recognition [6], and epidemiological modeling.

C. Experimental Data and Image Processing Utilities

For the experiments performed in this study, we acquired 25,000 raw image files produced by Canon EOS 40D and 350D cameras (10.1 and 8.0 megapixels, respectively). The entire raw dataset consumed roughly 350 GB of data on disk in Canon CR2 format. We used dcraw [7] to develop and convert the raw sensor data to viewable TIFF files, which required an additional 750 GB of disk space. dcraw also handled reading camera and lens-specific metadata from the files, which was stored alongside the raw images in Galileo.

Once developed and stored, additional *filters* that describe transformations on an image were applied using the ImageMagick [8] software suite. Available filters included blurring, sharpening, color normalization, modifying contrast and brightness, scaling, and applying color profiles. Each filter has a number of user-configurable parameters to control the impact of its transformation on the image, and these parameters were also stored in Galileo as metadata.

D. Test Environment

The system described in this paper was deployed on 25 HP DL-160 servers, each equipped with a quad-core Intel Xeon E5620 processor, 12 GB of RAM, and a 300 GB disk. The host operating system was Fedora 16 with OpenJDK 1.7.0.

III. Storage

Galileo is a distributed hash table, much like Dynamo [9] or Cassandra [10], but has a hierarchical layout consisting of collections of computing nodes called *groups*. The number of groups and the amount of nodes allocated to each group is a user-configurable setting in the system to best match computing demands. For our tests, we used five groups and allocated five nodes to each group.

To reduce latencies, Galileo attempts to place information with related features in the same groups, a strategy called

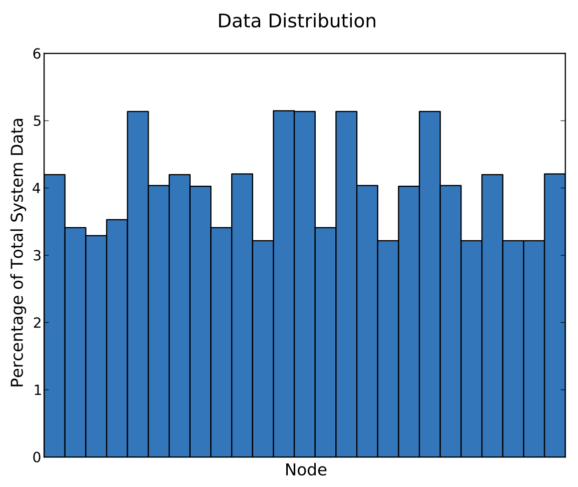

Fig. 2. Distribution of files in the system with our custom group hierarchy.

controlled dispersion. For this dataset, we grouped images based on the time reported by the camera when the pictures was taken. The remaining metadata in the files was concatenated and passed through an SHA-1 hash function to determine which node within a group would be responsible for each file. Figure 2 provides a visualization of how data was distributed across nodes in the system. While files are not completely evenly dispersed across the nodes, this storage imbalance allows the system to quickly locate images from particular photoshoots.

IV. Editing and Layers

After being developed, images often require editing to achieve the desired artistic effects. We store the original digital negative in Galileo, and then associate it with a number of editing steps, called *layers* that describe filters and transformations that can be applied to an image. These layers can be reordered or toggled at runtime, and serve as a set of processing steps that the system should take to produce a final image. Developed images produced by layers are also cached in the system as JPEG files of around 2 MB to speed up access times when users quickly browse through collections of images. To avoid changing the balance of storage load, the cache files (totalling about 50 GB) are stored alongside their raw counterparts. Table I provides timing information for the various processing workloads supported by layers in the system.

This test illustrates the computational complexity of developing digital negatives and applying image filters; on a quad-core desktop machine, our 25,000-image dataset would take roughly 7.5 days to process with all the layers applied. Another notable trend shown by this test is that most layers take around three seconds to process, which is likely due to the size of the images in our dataset. Newer SLR cameras would likely require 1.5x-3x more processing time. The order filters were applied in did not have a significant affect on processing time, but did produce different visual results.

TABLE I
Timing statistics for each processing workload. Each test was run 1,000 times. Up to four cores were used when possible.

	Time (sec)	**Std Dev (sec)**
Develop	3.4891	0.1416
Blur	3.5063	0.4807
Sharpen	10.3609	0.4708
Normalize	3.166	0.5395
Contrast	2.8422	0.4725
Brightness	2.6609	0.4252

V. Computation Benchmark

For our final test, we used the Galileo computation API to process layers in the Granules runtime across our entire dataset with different cluster sizes to illustrate the benefits of our distributed approach. Cluster sizes of 5, 10, 15, 20, and 25 nodes were used for this test. Figure 3 illustrates the improvement in performance as more nodes are added to the cluster. A sharp drop in the processing time is seen as the cluster grows from five nodes to ten, but diminishing returns are evident due to higher communication costs as more and more nodes are included in the cluster. Ultimately, though, the entire dataset can be processed in about 7.2 hours, rather than the 7.5 ***days*** needed on a modern desktop PC.

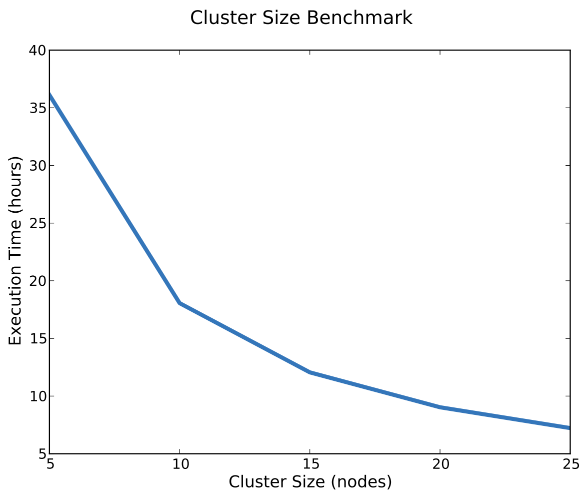

Fig. 3. Execution time as more processing elements (nodes) are added to the system.

VI. Conclusions and Future Work

A. Conclusions

This study has illustrated the viability of using a distributed computation engine and file system to process and manage large image datasets. Using the metadata features in Galileo, image modification history can be saved alongside the original raw sensor information to provide non-destructive editing capabilities. These features can also scale up to assimilate more computing resources and handle larger datasets from cameras with larger output sensor data.

B. Future Work

While the ability to apply a large set of filters directly to images is a key part of the modern photographer's post-processing workflow, additional creative modifications, such as spot retouching or airbrushing are also common editing features that we will support in the future. To add these capabilities, an image transformation would need to be associated with a path or points in an image and stored in Galileo. In addition, integrating our solution with a commercial photo editing suite would provide photographers with massive performance gains while also allowing them to use tools familiar to their field. Another possible avenue for expansion of this research is in the cloud, where users could upload and edit photos with a web interface backed by a large distributed pool of computing resources.

References

[1] M. Malensek, S. Pallickara, and S. Pallickara, "Galileo: A framework for distributed storage of high-throughput data streams," in *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on*, dec. 2011, pp. 17 –24.

[2] ——, "Exploiting geospatial and chronological characteristics in data streams to enable efficient storage and retrievals," *Future Generation Computer Systems*, 2012.

[3] ——, "Expressive query support for multidimensional data in distributed hash tables," in *(To appear) Utility and Cloud Computing (UCC), 2012 Fifth IEEE International Conference on*, 2011.

[4] S. Pallickara, J. Ekanayake, and G. Fox, "Granules: A lightweight, streaming runtime for cloud computing with support, for map-reduce," in *Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on*. IEEE, 2009, pp. 1–10.

[5] K. Ericson, S. Pallickara, and C. Anderson, "Analyzing electroencephalograms using cloud computing techniques," in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*. IEEE, 2010, pp. 185–192.

[6] ——, "Handwriting recognition using a cloud runtime," *Colorado Celebration of Women in Computing*, 2010.

[7] D. Coffin, "Dcraw," *Wikipedia online document*, 2008.

[8] L. ImageMagick Studio, "Imagemagick," 2008.

[9] Hastorun *et al.*, "Dynamo: amazons highly available key-value store," in *In Proc. SOSP*. Citeseer, 2007.

[10] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 2, pp. 35–40, 2010.