HIPI: A Hadoop Image Processing Interface for Image-based MapReduce Tasks

Chris Sweeney Liu Liu Sean Arietta Jason Lawrence

University of Virginia

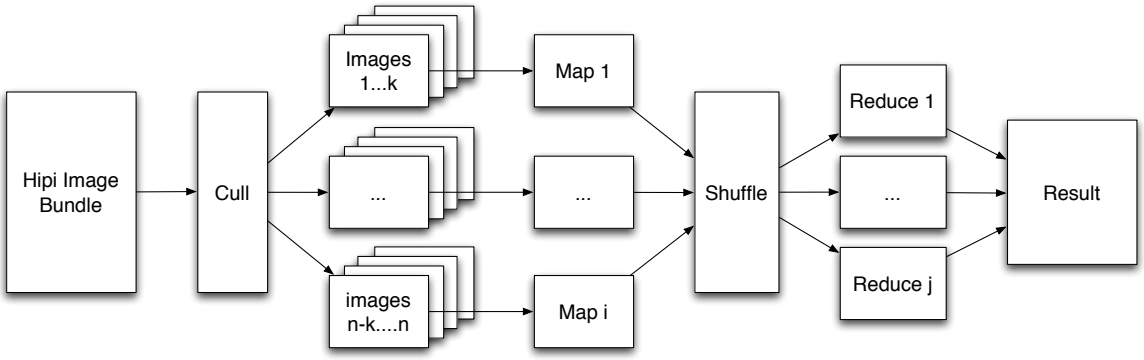

Figure 1: A typical MapReduce pipeline using our Hadoop Image Processing Interface with n images, i map nodes, and j reduce nodes

Abstract

The amount of images being uploaded to the internet is rapidly increasing, with Facebook users uploading over 2.5 billion new photos every month [Facebook 2010], however, applications that make use of this data are severely lacking. Current computer vision applications use a small number of input images because of the difficulty is in acquiring computational resources and storage options for large amounts of data [Guo... 2005; White et al. 2010]. As such, development of vision applications that use a large set of images has been limited [Ghemawat and Gobioff... 2003]. The Hadoop Mapreduce platform provides a system for large and computationally intensive distributed processing (Dean, 2004), though use of Hadoops system is severely limited by the technical complexities of developing useful applications [Ghemawat and Gobioff... 2003; White et al. 2010]. To immediately address this, we propose an open-source Hadoop Image Processing Interface (HIPI) that aims to create an interface for computer vision with MapReduce technology. HIPI abstracts the highly technical details of Hadoop's system and is flexible enough to implement many techniques in current computer vision literature. This paper describes the HIPI framework, and describes two example applications that have been implemented with HIPI. The goal of HIPI is to create a tool that will make development of large-scale image processing and vision projects extremely accessible in hopes that it will empower researchers and students to create applications with ease.

Keywords: mapreduce, computer vision, image processing

1 Introduction

Many image processing and computer vision algorithms are applicable to large-scale data tasks. It is often desirable to run these algorithms on large data sets (e.g. larger than 1 TB) that are currently limited by the computational power of one computer [Guo... 2005]. These tasks are typically performed on a distributed system by dividing the task across one or more of the following features: algorithm parameters, images, or pixels [White et al. 2010]. Performing tasks across a particular parameter is incredibly parallel and can often be perfectly parallel. Face detection and landmark classification are examples of such algorithms [Li and Crandall... 2009; Liu et al. 2009]. The ability to parallelize such tasks allows for scalable, efficient execution of resource-intensive applications. The MapReduce framework provides a platform for such applications.

Basic vision applications that utilize Hadoops MapReduce framework require a staggering learning curve and overwhelming complexity [White et al. 2010]. The overhead required to implement such applications severely cripples the progress of researchers [White et al. 2010; Li and Crandall... 2009]. HIPI removes the highly technical details of Hadoops system and provides users with the familiar feel of an image library with the access to the advanced resources of a distributed system [Dean and Ghemawat 2008; Apache 2010]. Our platform is focused around giving users unprecedented access to image-based data structures with a pipeline that is intuitive to the MapReduce system, allowing for easy and flexible use for vision applications. Because of the similar goals in our frameworks, we take particular inspiration from the Frankencamera project as a model for designing an open API to provide access to computing resources.

We have designed HIPI with the aims of providing a platform specific enough to contain a relevant framework applicable for all image processing and computer vision applications but flexible enough to withstand continual changes and improvements within Hadoops Mapreduce system. We expect HIPI to promote vision research for these reasons. HIPI is largely a software design project, driven by the overarching goals of abstracting of Hadoop's functionality into an image-centric system and providing an extendible

system that will provide researchers with a tool to effectively use Hadoops Mapreduce system for image processing and computer vision. We believe that this ease of use and user control will make the process for creating large-scale vision experiments and applications. As a result, HIPI serves as an excellent tool for researchers in computer vision because it allows development of large-scale computer vision applications to be more accessible than ever. To our knowledge, we are the first group to provide an open interface for image processing and computer vision applications for Hadoops Mapreduce platform [White et al. 2010].

In the following section, we will describe previous work in this area. In particular, we discuss the motivation for creating an image-based framework that allows large-scale vision applications. Next, we describe the the overview for the HIPI library including the cull, map, and reduce stages. Additionally, describe our approach for distributing tasks for the MapReduce pipeline. Finally, we demonstrate the capabilities of HIPI with two examples of vision applications using HIPI.

2 Prior Work

With the proliferation of online photo storage and social medias from websites such as Facebook, Flickr, and Picasa, the amount of image data available is larger than ever before and growing more rapidly every day [Facebook 2010]. This alone provides an incredible database of images that can scale up to billions of images. Incredible statistical and probabilistic models can be built from such a large sample source. For instance, a database of all the textures found in a large collection of images can be built and used by researchers or artists. The information can be incredibly helpful for understanding relationships in the world[1]. If a picture is worth a thousand words, we could write an encyclopedia with the billions of images available to us on the internet.

These images are enhanced, however, by the fact that users are supplying tags (of objects, faces, etc.), comments, titles, and descriptions of this data for us. This information supplies us with an amazing amount of unprecedented context for images. Problems such as OCR that remain largely unsolved can make bigger strides with this available context guiding them. Stone et al. describe in detail how social networking sites can leverage facial tagging features to significantly enhance facial recognition. This idea can be applied to a wider range of image features that allow us to examine and analyze images in a revolutionary way.

It is these reasons that motivate the need for research with vision applications that take advantage of large sets of images. MapReduce provides an extremely powerful framework that works well on data-intensive applications where the model for data processing is similar or the same. It is often the case with image-based operations that we perform similar operations throughout an input set, making MapReduce ideal for image-based applications. However, many researchers find it impractical to be able to collect a meaningful set of images relevant to their studies [Guo... 2005]. Additionally, many researchers do not have efficient ways to store and access such a set of images. As a result, little research has been performed on extremely large image-sets.

[1] One can imagine that applications such as object detection could provide information that enable researchers to recognize the relationships between certain objects (e.g. bumblebees are often in pictures of flowers). There are many examples of useful applications such as these.

3 The HIPI Framework

HIPI was created to empower researchers and present them with a capable tool that would enable research involving image processing and vision to be performed extremely easily. With the knowledge that HIPI would be used for researchers and as an educational tool, we designed HIPI with the following goals in mind.

1. Provide an open, extendible library for image processing and computer vision applications in a MapReduce framework
2. Store images efficiently for use in MapReduce applications
3. Allow for simple filtering of a set of images
4. Present users with an intuitive interface for image-based operations and hide the details of the MapReduce framework
5. HIPI will set up applications so that they are highly parallelized and balanced so that users do not have to worry about such details

3.1 Data Storage

Hadoop uses a distributed file system to store files on various machines throughout the cluster. Hadoop allows files be accessed, however, without knowledge of where it is stored in the cluster, so that users can reference files the same way they would on a local machine and Hadoop will present the file accordingly.

When performing MapReduce jobs, Hadoop attempts to run Map and Reduce tasks at the machines were the data being processed is located so that data does not have to be copied between machines [Apache 2010]. As such, MapReduce tasks run more efficiently when the input is one large file as opposed to many small files[2]. Large files are significantly more likely to be stored on one machine whereas many small files will likely be spread out among many different machines, which requires significant overhead to copy all the data to the machine where the Map task is [Ghemawat and Gobioff... 2003]. This overhead can slow the runtime ten to one hundred times [White 2010]. Simply put, the MapReduce framework operates more efficiently when the data being processed is local to the machines performing the processing.

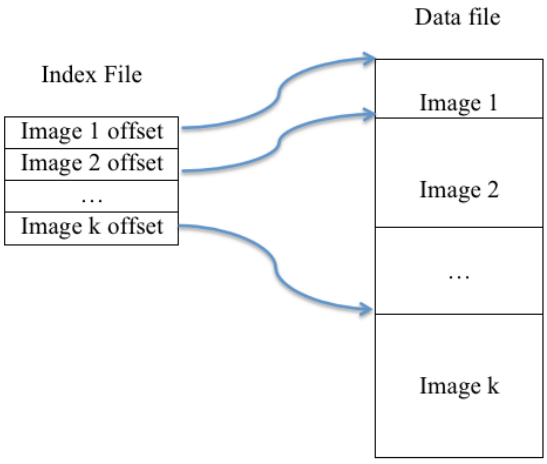

Figure 2: A depiction of the relationship between the index and data files in a HIPI Image Bundle

[2] Small files are files that are considerably smaller than the file block size for the machine where the file resides

With this in mind, we created a HIPI Image Bundle data type that stores many images in one large file so that MapReduce jobs can be performed more efficiently. A HIPI Image Bundle consists of two files: a data file containing concatenated images and an index file containing information about the offsets of images in the data file as shown in Figure-2. This setup allows us to easily access images across the entire bundle without having to read in every image.

We observed several benefits of the HIPI Image Bundle in tests against Hadoop's Sequence file and Hadoop Archive (HAR) formats. As White et. al. note, HARs are only useful for archiving files (as backups), and may actually perform slower than reading in files the standard way. Sequence files perform better than standard applications for small files, but must be read serially and take a very long time to generate. HIPI Image Bundle have similar speeds to Sequence files, do not have to be read serially, and can be generated with a MapReduce program [White 2010; Conner 2009]. Additionally, HIPI Image Bundles are more customizable and are mutable, unlike Sequence and HAR files. For instance, we have implemented the ability to only read the header of an image file using HIPI Image Bundles, which would be considerably more difficult with other file types. Further features of the HIPI Image Bundles are highlighted in the following section

3.2 Image-based MapReduce

Standard Hadoop MapReduce programs handle input and output of data very effectively, but struggle in representing images in a format that is useful for researchers. Current methods involve significant overhead to obtain standard float image representation. For instance, to distribute a set of images to a set of Map nodes would require a user to pass the images as a String, then decode each image in each map task before being able to do access pixel information. This is not only inefficient, but inconvenient. These tasks can create headaches for users and make the code look cluttered and difficult to interpret the intent of the code. As such, sharing code is less efficient because the code is harder to read and harder to debug. Our library focuses on bringing familiar image-based data types directly to the user for easy use in MapReduce applications.

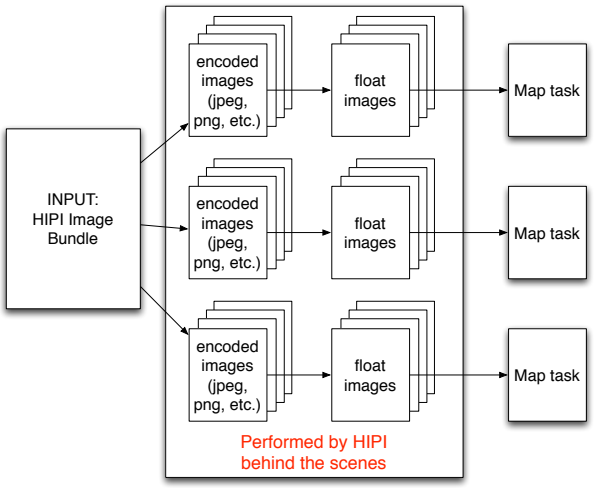

Figure 3: The user only needs to specify a HIPI Image Bundle as an input, and HIPI will take care of parallelizing the task and sending float images to the mappers

Using the HIPI Image Bundle data type as inputs, we have created an input specification that will distribute images in the HIPI Image Bundle across all map nodes. We distribute images such that we attempt maximize locality between the mapper machines and the machine where the image resides. Typically, a user would have to create InputFormat and RecordReader classes that specify how the MapReduce job will distribute the input, and what information gets sent to each machine. This is task is nontrivial and often becomes a large point of headaches for users. We have included InputFormat and RecordReaders that take care of this for the user. Our specification works on HIPI Image Bundles for various image types, sizes, and varying amounts of header and exif information. We handle all of these different image permutations behind the scenes to bring images straight to the user as float images. No work is needed to be done by the user, and float images are brought directly to the Map tasks in a highly parallelized fashion.

During the distribution of inputs but before the map tasks start we introduce a culling stage to the MapReduce pipeline. The culling stage allows for images to be filtered based on image properties. The user specifies a culling class that describes how the images will be filtered (e.g. pictures smaller than 10 megapixels, pictures with GIS location header data). Only images that pass the culling stage will be distributed to the map tasks, preventing unnecessary copying of data. This process is often very efficient because culling often occurs based on image header information, so it is not required to read the entire image.

Additionally, images are distributed as float images so that users can immediately have access to pixel values for image processing and vision operations. Images are always stored as standard image types (e.g. JPEG, PNG, etc.) for efficient storage, but HIPI takes care of encoding and decoding images to present the user with float images within the MapReduce pipeline. As a result, programs such as calculating the mean value of all pixels in a set of images can be written in merely lines. We provide operations such as cropping for image patches extraction. It is often desirable to access image header and exif information without need for pixel information, so we have abstracted this information from the pixel data. This is particularly useful for the culling stage, and for applications such as im2gis[3] that need access to metadata. Presenting users with intuitive interfaces for accessing data relevant to image processing and vision applications will allow for more efficient creation of MapReduce applications.

4 Examples

We describe two non-trivial applications performed using the HIPI framework to create MapReduce jobs. These applications are indicative of the types of applications that HIPI enables users concerned with large-scale image operations to easily do. These examples are difficult and inefficient on existing platforms, yet simple to implement with the HIPI API.

4.1 Principal Components of Natural Images

As an homage to Hancock et. al, we computed the first 15 principal components of natural images. However, we decided instead of randomly sampling one patch from 15 images, we sampled over 1000 images and 100 patches in each. The size of our input set was 10,000 times larger than the original experiment. Additionally, we did not limit our images to natural images like the original experiment (though we could do this in the cull stage). As such, results differ but hold similar characteristics.

We parallelize the process of computing the covariance matrix for the images according to the following formula, where x_i is a sample

[3] http://graphics.cs.cmu.edu/projects/im2gps/

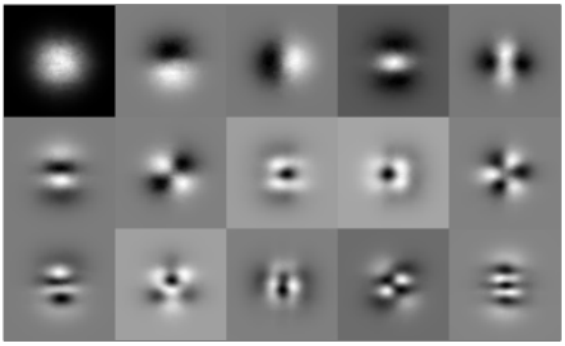

Figure 4: The first 15 principal components of 15 randomly sampled natural images, as observed by Hancock et al. from left to right, top to bottom

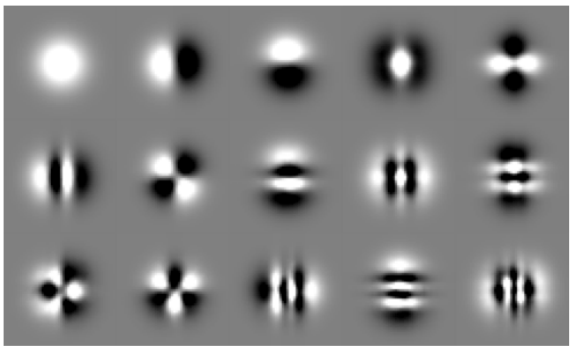

Figure 5: The first 15 principal components of 100,000 randomly sampled image patches, as calculated with HIPI from left to right, top to bottom

patch and $\bar{x}$ is the sample patch mean

$$COV = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})(x_i - \bar{x})^T \quad (1)$$

Equation-1 suits HIPI perfectly because the summation is grossly parallel. In other words, we can easily compute each term in the summation independently (assuming we already have the mean), thus, we can compute each term in parallel. We first run a MapReduce job to compute the sample mean, then use that as $\bar{x}$ for future covariance calculation. Then, we run a MapReduce job that computes $(x_i - \bar{x})(x_i - \bar{x})^T$ for all 100 patches from each image[4]. Because HIPI allocates one image per map task, it is simple to randomly sample an image for 100 patches and perform this calculation. Each map task will then emit the summation of its partial covariances sampled from the image to the reducer, where all partial covariances will be summed to calculate the covariance for the entire sample set.

After determining the covariance for 100,000 randomly sampled patches, we used Matlab to find the first 15 principal components. As expected, images do not correlate perfectly because we are using far different inputs to our experiments, and our display of positive and negative values also may differ slightly. However, certain principal components are the same (1, 7, 12), are merely switched (2 and 3, 4 and 5), or show some resemblance to the original experiment (15). Performing a principal component analysis on a massive, unrestricted data set gives us unparalleled knowledge about images. For tasks such as these, HIPI excels.

[4]We call this partial sum the partial covariance

4.2 Downloading Millions of Images

Step 1: Specifiy a list of images to collect. We assume that there exists a well-formed list containing url's of images to download. This list should be stored in a text file with exactly one image url per line. This list can be generated by hand, from MySQL, or from a search query (e.g. google images, flickr, etc.). In addition to the list of images, the user will input the number of nodes to run the task. According to this input, we divide the image set across the specified number of nodes for maximum efficiency and parallelism when downloading the images. Each node in the Map task will generate a HIPI Image Bundle containing all of the images it downloaded, then the Reducer will merge all the HIPI Image Bundles together to form one large HIPI Image Bundle.

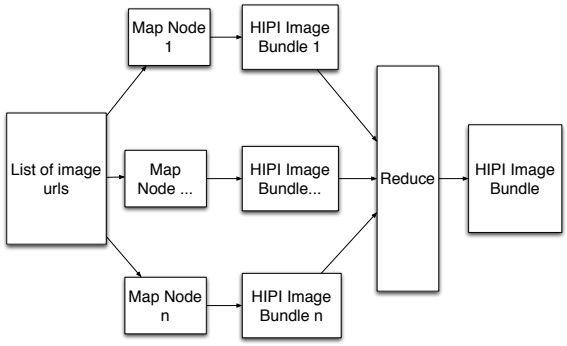

Figure 6: A demonstration of the parallelization in the Downloader application. The task of downloading the list of images urls is split amongst n map tasks. Each mapper creates a HIPI Image Bundle, which is merged into one large HIPI Image Bundle in the Reduce phase

Step 2: Split URLs into groups and send each group to a Mapper. Using the inputted list of image urls and the number of nodes used to download these images, we equally distribute the task of downloading images to the specified number of map nodes. This allows for maximum parallelization for the downloading process. Image urls are been distributed to the various nodes equally, and the map tasks will begin downloading each image in the set of urls it is responsible for, as Figure-6 shows.

Step 3: Download images from the internet. We then establish a connection to the url retrieved from the database and download the image using java's URLConnection class. Once connected, we check the file type to make sure it is a valid image, and get an InputStream to the connection. From this, we can use the InputStream to add the image to a HIPI Image Bundle.

Step 4: Store images in a HIPI Image Bundle. Once the InputStream is received from the URLConnection, we can add the image to a HIPI Image Bundle simply by passing the InputStream to the addImage method. Each map task will then generate a HIPI Image Bundle, and the Reduce phase will merge all of the bundles together into one large bundle.

By storing images this way, you are able to take advantage of our HIPI framework for MapReduce tasks that you want to perform on image sets at a later point. For example, to check the results of the Downloader program, we ran a very simple MapReduce Program (7 lines) that was able to take the HIPI Image Bundle and write out the images to individual JPEG files on the HDFS effortlessly.

5 Conclusion

This paper has described our library for image processing and vision applications on a MapReduce framework - Hadoop Image Processing Interface (HIPI). This library was carefully designed to hide the complex details of Hadoop's powerful MapReduce framework and bring to the forefront what users care about most: images. Our system has been created with the intent to operate on large sets of images. We provide a format for storing images for efficient access within the MapReduce pipeline, and simple methods for creating such files. By providing a culling stage before the mapping stage, we give the user a simple way to filter image sets and control the types of images being used in their MapReduce tasks. Finally, we provide image encoders and decoders that run behind the scenes and work to present the user with float image types which are most useful for image processing and vision applications.

Through these features, our interface brings about a new level of simplicity for creating large-scale vision applications with the aim of empowering researchers and teachers with a tool for efficiently creating MapReduce applications focused around images. This paper describes two example applications built with HIPI that demonstrate the power it presents users with. We hope that by bring the resources and power of MapReduce to the vision community that we will enhance the ability to create new vision projects that will enable users to push the field of computer vision.

6 Acknowledgements

We would like to give particular thanks to PhD Candidate Sean Arietta for his guidance and mentoring throughout this project. His leadership and vision have been excellent models and points of learning for us throughout this process. Additionally, we must give great thanks to Assistant Professor Jason Lawrence for his support throughout the past several years and for welcoming us into UVa's Graphics Group as bright eyed undergraduates.

References

ADAMS, A., JACOBS, D., DOLSON, J., TICO, M., PULLI, K., TALVALA, E., AJDIN, B., VAQUERO, D., LENSCH, H., AND HOROWITZ, M. 2010. The frankencamera: an experimental platform for computational photography. *ACM SIGGRAPH 2010 papers*, 1–12.

APACHE, 2010. Hadoop mapreduce framework. http://hadoop.apache.org/mapreduce/.

CONNER, J. 2009. Customizing input file formats for image processing in hadoop. *Arizona State University. Online at: http://hpc. asu. edu/node/97*.

DEAN, J., AND GHEMAWAT, S. 2008. Mapreduce: Simplified data processing on large clusters. *Communications of the ACM 51*, 1, 107–113.

FACEBOOK, 2010. Facebook image storage. http://blog.facebook.com/blog.php?post=206178097130.

GHEMAWAT, S., AND GOBIOFF..., H. 2003. The google file system. *ACM SIGOPS Operating ...* (Jan).

GUO..., G. 2005. Learning from examples in the small sample case: face expression recognition. *Systems* (Jan).

HANCOCK, P., BADDELEY, R., AND SMITH, L. 1992. The principal components of natural images. *Network: computation in neural systems 3*, 1, 61–70.

LI, Y., AND CRANDALL..., D. 2009. Landmark classification in large-scale image collections. *Computer Vision* (Jan).

LIU, K., LI, S., TANG, L., AND WANG..., L. 2009. Fast face tracking using parallel particle filter algorithm. *Multimedia and Expo* (Jan).

STONE, Z., AND ZICKLER..., T. 2010. Toward large-scale face recognition using social network context. *Proceedings of the IEEE* (Jan).

WHITE, B., YEH, T., LIN, J., AND DAVIS, L. 2010. Web-scale computer vision using mapreduce for multimedia data mining. *Proceedings of the Tenth International Workshop on Multimedia Data Mining*, 1–10.

WHITE, 2010. The small files problem. http://www.cloudera.com/blog/2009/02/the-small-files-problem/.