Approximate Algorithmic Image Matching to Reduce Online Storage Overhead of User Submitted Images

Braden D. Licastro
Department of Computer Science
Allegheny College
licastb@allegheny.edu
http://skynetgds.no-ip.biz/srthesis

October 28, 2013

Abstract

Websites similar to Photobucket and imgur must be able to store massively increasing quantities of data; most of these sites already implement systems that limit the number, size, type of image, and compress the uploaded images to save space. Though these systems can be paired with file expiration times in which an upload is deleted, the cost of storage and backups of this data can be high. To further address this problem, the research aims to create an intelligent image uploading system capable of identifying near-duplicate image uploads on-the-fly; This system also provides the user with a higher resolution copy of their image from the server if available, thus providing better user experience while reducing unnecessary redundancy. By using the proposed system, the user will benefit from improved quality and service while the business can reduce storage and other various costs.

1 Introduction

Computers are absolutely everywhere, and society is becoming more digitized every day creating a need for storage of large amount of files. When running a website, storage space is at a premium, and the physical disks in the servers being used may not be local, but may in fact be located around the world. By allowing user submitted data, it quickly becomes apparent that data redundancy reduction can save a significant amount of storage space. This research will target the problem of data redundancy and provide a solution that will reduce not only storage costs but also inefficiencies. An implementation of a file management system will be introduced that is capable of managing large volumes of files using a database of location pointers and hash keys that identify each individual file. By using pointers, it is possible for the files to be stored on one central location or distributed across several systems while avoiding long search times. By using text based entries in a database, a hashing function can be implemented which is capable of producing a unique hash code for each file in the collection. Because this key is unique for each different file, it is a reliable gauge of uniqueness. When multiple files are found to be identical only one physical copy will be kept but pointers to the database entry will allow easy access. On large, distributed network storage servers, the benefits become ever more apparent. As users store files on the disks, the system will hash each file and search for a match in a database, thus allowing the user access to the data, but eliminating the overhead of

storing numerous copies of identical data. An increase in computation time will exist but should be minimal as text comparisons will be the primary task in finding uniqueness, but this will be outweighed by minimized physical storage requirements and costs of data backup. The implementation of this system could be used in tandem with other currently implemented methods of file space cost reduction technologies out of the box. By not only using the proposed method of space requirement reduction, it could in theory be possible that with data expiration times, file size maximums, and similar technologies, that storage needs will plateau after the initial surge of additions and the expiration time has passed for the earliest uploaded file. This would allow webmasters to better predict overall needs and better predict upload trends and react accordingly with preventative maintenance and any required space additions.

2 Related Work

Though I will not be comparing existing images located in a database, the paper by Lee, Ke, and Isard [7] can be useful in method of approach. During their research they utilized a Partition min-hash function for discovering near-duplicate images. Their algorithm doesn't look at the image as a whole as many other algorithms do, but it in fact only looks at the image in several smaller pieces. They claimed that the algorithm is actually faster and more effective than standard min-Hash functions which look at the entire image as they only require several smaller regions. Since the proposed research I will be performing is highly efficiency sensitive due to its web based structure, I believe this method of image comparison would be highly valuable and will attempt to reconstruct the algorithm in one manner or another.

Another article that will be vital to creating an effective an efficient image matching algorithm will be the 2010 paper by Srinivasan. This research was target at web-based image matching, which is exactly what I am going to perform. This research claims that traditional near-duplicate detection systems are not applicable for the de-duplication of large-scale web image collections. [10] The research performed targeted an image matching system which was scalable, highly efficient, and effective.

$$F_\sigma(k,v) = \frac{1}{2\pi}\int_0^1 \int_0^{2\pi} f(r,\theta) r^{\sigma - iv} e^{-ik\theta} \frac{dr}{r} d\theta$$

Figure 1: Fourier Transform Used by Authors

In order to perform the effective image matching the authors required, they decided to implement a thumbnail matching based system. This algorithm would generate a 130-bit thumbnail representation of the image using the Fourier Transform method outlined in figure 1 and was capable of finding near-duplicate images while operating in $O(1)$ time. [10]

When using this thumbnail method, the authors were able to adjust automatically for differences in resolution, arbitrary amounts of cropping, caption, logo, and other manipulations, in assition to color, and rotation variations. This will all be vitally important with the research being proposed, as each of these are highly probable, and will likely each be included in the final tests performed on the implemented system.

Due to the web based nature of the proposed research, another article, written by Foo, Zobel, Sinha, and Tahaghoghi in 2007 focuses primarily on web search based image matching. Their research primarily targets the finding and determination of types of copyright infringement as most near-duplicate images are derived from one source. The goal of their research was not to derive an algorithm capable of matching images, but was to locate duplicate and near-duplicate images on the web using search engines, and determine the most common methods of image alteration leading to redundancy and possibly copyright infringement. Their research will be invaluable to the proposed research as it will then be possible to target the most common methods of image alteration when creating a matching algorithm. To locate their image set to work with, they used the most popular search queries of 2005 and collected the number of images, and determined the number of unique images based off of the results returned. As a note, images with non-unique URLs were not included to prevent false duplicates from the same sites. In conclusion, the authors found that the most common alterations, ordered from one to 10, one being the most common were as follows: [4]

1. **Combination:** Images with one or more alterations.
2. **Scale:** Images that difer in size.
3. **Crop/Zoom:** Images that are cropped from the original.
4. **Picture in Picture:** Images that contain another image.
5. **Contract:** Images that have adjusted contrast.
6. **Border/Frame:** Images that have added borders or frames.
7. **Grayscale:** Images that have been converted to grayscale.
8. **Recoloring:** Images with colors that have been modified.
9. **Mirror:** Images that have been mirrored to prevent copyright infringement detection.
10. **Rotate:** Images that have been rotated.

The research proposed will not focus on every one of these common parameters that have been found, but will focus on a select group of these common alterations outlined in section 3.

Finally, the most relevant information found will provide a codebase for the algorithm and research to springboard from. An individual outlined a method of using PHP libraries to generate real-time thumbnails and hashes of images and compare them to ones currently located on a server before uploading. If the image is a duplicate it would be denied the privilege of being uploaded and tell the user so. This system is extremely limited as it provides the user with no way of adding an image to the server if it is not a duplicate and is only a false positive, and if it is unique, does not provide them with a method of accessing the already-present file.

The code used to generate the image will be used as a base to generate a fully functional image sharing site with duplicate-reduction systems and allow for the testing of the effectiveness of such a system over traditional methods of sharing with duplicates allowed. The system created

by the author generates a 16x16 thumbnail of each image on the server, and of the image being uploaded. From here, the algorithm will compare the thumbnail, black and white, and color histograms, and determine if it is a duplicate based off the threshold setting set by the server administrator for allowable differences. [2]. The proposed research will utilize this exact method, but will provide code improvements to utilize less memory by implementing the PHP Improved libraries and also checking images for immediate similarities in resolution, exact image matching with a full MD5 image hash, and by comparison of metadata. These methods require minimal calculation and is a simple comparison of strings compared to the generation of histograms and resizing of images as a first step. After the initial string comparison approach, the provided code will be used to calculate the average deviation between the teo images and will allow for easy expansion of the system to hopefully provide a faster more effective image matching system.

Using the remainder of the section to generate the bibliography that I didn't reference. [8] [1] [6] [3] [9] [5]

3 Method of Approach

In order to implement the proposed research, the website will be run on a custom server with an allocated 5 Gigabytes of dedicated storage, 8 Gigabytes of RAM, and a Core i3 2.43 GHz processor. All services and programs will be shut down during the experiment except for Apache, MySQL, PHP, and Webmin to ensure consistent results. Throughout the duration of the experiment, no software updates will be installed other than security related operating system updates. The website must be designed to be lightweight, have no extraneous scripts or applets running on the upload page, and will be compatible with WebKit browsers such as Firefox. This limitation will allow for a focused effort in file management on a specific platform and will allow room for further research after the tool has been optimized. To start, a database will be implemented so that it is able to hold a hashed key for each uploaded image and the location of the file on the disk in addition to upload statistics such as upload method and time taken to complete the task. An example of this schema can be seen in (schema to be added when finalized).

The website will run multiple scripts for each upload. The first script will be a traditional upload method where the user supplies a file, it will be uploaded to the server, and the server will return a unique URL to access the image from at a later time. During this process, each file will be assigned a new, unique name to prevent collisions upon upload. An INSERT will then be run on the database for the image and will record the files location on the server, the file name, that it was uploaded using the traditional method, the unique URL to access it from, and an ID for each insert into the database. After the completion of the process an UPDATE will be run on the database entry created by the INSERT above. The time taken to complete the task will be included in that entry and can be used to analyze the efficiency of both systems upon the completion of the tests outlined in 4.

After the first script completes, a second script will be called. This will perform mostly the same function as the traditional upload script, but will check each upload for duplicates and handle each case appropriately. This script will use the file uploaded and used in the section above. The image will be taken and converted into a 16x16 black-and-white thumbnail and a histogram will be produced from that image. The server will then look into the database and check whether or not any images uploaded with the duplicate reduced system are similar to the one looking to be

uploaded. If there are no matches, then the image will be uploaded to the server. At this time an INSERT will then be run on the database for the image and will record the files location on the server, the file name, that it was uploaded using the duplicate-reduced method, the unique URL to access it from, the image's MD5 hashed histogram that is described below, and an ID for each insert into the database. After the completion of the process an UPDATE will be run on the database entry created by the INSERT above.

If the image histogram closely matches or is an identical match to the image on the server, a color profile will be generated real time for both images and compared. If there is a match or close match, the script will compare several small, randomly chosen blocks on the images. If these match, then the image is assumed to be a duplicate and a prompt will be displayed to the user showing them the image they provided and the possible match that already exists. If the image is verified a match, the higher of the two resolutions will be kept, and the user will be given a unique link to the image. If the user decides the image is not a duplicate, the system will run an INSERT on the database as outlined earlier, and it will be added to the database. Following the completion of this process an UPDATE will be run on the lase INSERT and the time taken to complete the task will be included in that entry.

In the case of requiring duplicate files, where the user decides to upload the image regardless of uniqueness, the script will be able to differentiate between the two images with the unique image ID that is generated at the time of insertion into the database. The first occurrence of the duplicate image or the one closest matching a new upload will be used in the prompt to the user to prevent frustration with multiple prompts every time more than one duplicate is found.

If an image is indeed found to be duplicate, and the user chooses to use the higher resolution image that is on the server, a unique link will be generated for them and an INSERT statement will be run containing the same information as the matching file's insert did at the time of upload. This will allow multiple links that point to the same image and the user will be unaware of the system operating in the background which will provide a consistent experience. An outline of this full process can be seen in figure 2

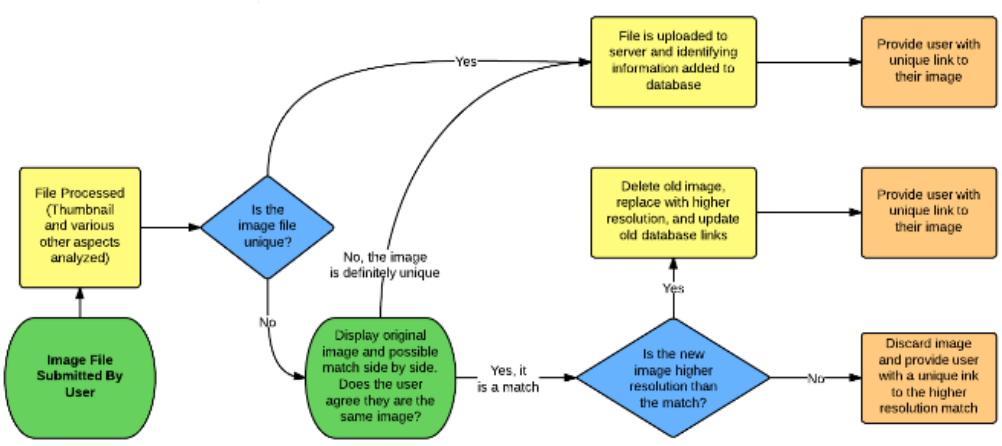

Figure 2: Streamlined upload process

When a user accesses the file from the provided link, the system will run a query that looks up the image identifier provided in the URL, if it matches an image on the server, the image location will be used to provide the image to the user for viewing. The user never notices a difference, but on the server side we have ensured file redundancy has been eliminated and possibly improved user experience by providing the user with higher quality content than what they were expecting. If the image is not found, a 404 "Image cannot be found." error will be displayed to let the user know something went wrong.

4 Evaluation Strategy

Evaluating the effectiveness of the website will entail several benchmark readings. The total number of files, number of unique files, and the overall size of the benchmark folder versus the duplicate reduced upload folder will all be used in determining the effectiveness of the image comparison algorithms implemented in the website. To perform the tests, a fixed set of images will be used. These images will consist of a collection of 15 artificially generated gray-scale, 15 natural gray-scale, and 15 natural RGB images in addition to 5 duplicate images in each category. These duplicates will consist of one of each of the following images: identical, diminished resolution, text overlaid, reduced resolution, and a color/contrast correction.

$$\left(\frac{Base - Reduced}{Base}\right) * 100 = \%\ Improvement\ over\ Base$$

Figure 3: Equation for determining efficiency improvements

In order to test the algorithm, all 15 unique images will be uploaded to the server using the built in upload function. Any false positive matches will be noted and the algorithm will be adjusted until none remain. At this point, the baseline folder and the duplicate reduced folders should contain the same set of data. Next, the duplicate images will be uploaded to the server using the same method. The system should find all remaining images as duplicates and prompt the user with the option to upload anyway or use the existing image. For the test I will not allow the upload. After all images have been uploaded, the total number of files in the baseline directory will be compared to the number of files in the duplicate reduced directory using the algorithm in 3, where base is the baseline directory number to be compared and reduced is the duplicate reduced directory number to be compared. The directory sizes and average upload times will also be evaluated using the algorithm in 3. The evaluated result, if positive, will denote a positive improvement in efficiency compared to the baseline directory, and a negative result is interpreted as a decrease in efficiency over the baseline directory.

After analyzing the results of the test outlined above, the algorithm will be refined to provide more accurate results and the folders and database will be reset. At this point, the tests will be run again, and any changes in effectiveness will be noted. Because the system implements a text-based database, the size difference between the baseline and duplicate reduced will be negligible as no image specific information is stored. In addition, the number of entries will be identical due to the upload process outlined in 3, which generates a unique URL for every image upload request whether the image is uploaded or an existing one on the server is used.

5 Research Schedule

The research schedule for the project is to be tentatively based off of the following deadlines. Due to the nature of the project and the wide array of pre-existing software required needing to be modified and implemented, unseen hurdles may arise.

Phase 1 - 1 month: Find acceptable operating system set up a fully functional LAMP server to host website and provide storage of submitted images. During this time, I will begin learning and installing PHP and JavaScript libraries so code implementation can begin shortly after the completion of server setup.

Phase 2 - 2-3 months: Develop a working image sharing website which is capable of satisfying the requirements outlined in 3. Set up two basic file and database structures to be used throughout the experiment. One will be used as a baseline and the other structure will function as the duplicate-reduced system. Both will host the image files, file information, and time taken to add files to the database. The system will also track the average upload time for each method, the total number of files, and the total size of each upload directory.

Phase 3 - 1 week: During this time I will run various tests on the website using the proposed methods in 3 and record the results of the experiment. After the new image files have been added to the collection, some unique, and others duplicate, the submissions directory on the server will be scanned for duplicates and ensure none were accidentally added. Results to be recorded are the final number of files, the number of duplicate images, and the total storage space used.

Phase 4 - remaining time: The results gathered throughout the test process will be evaluated and the website algorithms tweaked and finalized. After modifications are complete tests will be rerun and the final results will be evaluated.

6 Conclusion

With computers becoming ever more prevalent and data being generated and shared at a rapidly increasing rate, it is becoming more and more necessary every day to reduce the redundancy of shared data and create more efficient sharing systems. By removing duplicate image files from sharing sites and preventing the addition of new duplicates we can begin to tackle this issue and begin reducing the storage requirements of running such systems. Aside from just reducing storage overhead it can also be possible to reduce time overhead of searching for images and the amount of bandwidth needed to operate these systems. In addition, a significant amount of money can be saved as a result of reduced electrical costs per node, and even reduce the number of server nodes in extreme cases of redundancy.

References

[1] Angela Bradley. Renaming PHP Uploads. http://php.about.com/od/advancedphp/ss/rename_upload.htm, 2011. [Online; accessed 10-October-2013].

[2] CatPa.ws. PHP GD Duplicate Image Finder. http://www.catpa.ws/php-duplicate-image-finder/, 2010. [Online; accessed 13-September-2013].

[3] Ubuntu Documentation. ApacheMySQLPHP - LAMP Server Setup Guide. http://php.net/manual/en/ref.image.php, 2013. [Online; accessed 21-August-2013].

[4] Jun Jie Foo, Justin Zobel, Ranjan Sinha, and S. M. M. Tahaghoghi. Detection of Near-duplicate Images for Web Search. In *Proceedings of the 6th ACM International Conference on Image and Video Retrieval*, CIVR '07, pages 557–564, New York, NY, USA, 2007. ACM.

[5] The PHP Group. GD and Image Functions. http://php.net/manual/en/ref.image.php, 2013. [Online; accessed 13-October-2013].

[6] The PHP Group. POST Method Uploads. http://de.php.net/manual/en/features.file-upload.post-method.php, 2013. [Online; accessed 09-September-2013].

[7] David C. Lee, Qifa Ke, and Michael Isard. Partition Min-hash for Partial Duplicate Image Discovery. In *Proceedings of the 11th European Conference on Computer Vision: Part I*, ECCV'10, pages 648–662, Berlin, Heidelberg, 2010. Springer-Verlag.

[8] Dejan Marjanovic. PDO vs. MySQLi: Which Should You Use? http://net.tutsplus.com/tutorials/php/pdo-vs-mysqli-which-should-you-use/, 2012. [Online; accessed 02-October-2013].

[9] nixCraft. Ubuntu Linux: Install or Add PHP-GD Support to Apache Web Server. http://www.cyberciti.biz/faq/ubuntu-linux-install-or-add-php-gd-support-to-apache/, 2012. [Online; accessed 30-September-2013].

[10] S H. Srinivasan and Neela Sawant. Finding Near-duplicate Images on the Web Using Fingerprints. In *Proceedings of the 16th ACM International Conference on Multimedia*, MM '08, pages 881–884, New York, NY, USA, 2008. ACM.