CStorage: An Efficient Classification-based Image Storage System in Cloud Datacenters

Haiying Shen
Department of Computer Science
University of Virginia
Charlottesville, VA
Email: hs6ms@virginia.edu

Heng Zhou
School of Computing
Clemson University
Clemson, SC
Email: hzhou3@clemson.edu

***Abstract*—Image storage systems are designed to support images sharing and retrieving applications, especially for smartphone and other mobile devices. Existing image storage systems store images in random data servers without considering their similarities. When a front-end server sends out an image query, it receives similar images from a large number of data servers, causing possible network incast congestion and long query latency. To solve this problem, we propose CStorage, an efficient classification-based image storage system. In CStorage, similar images are stored in the same data server. Thus, a front-end server receives query results from a single data server, which reduces the occurrence of incast congestion as well as the image retrieval latency. CStorage also leverages the deep learning technique to cluster images and provides a high precision and recall rates. Experimental results show the effectiveness of CStorage in reducing image retrieval latency and improving precision and recall rates of searching results.**

***Index Terms*—Image storage systems, Locality-sensitive hashing, Deep learning, Incast congestion**

I. Introduction

With the development of mobile device technology and popularity of mobile devices, taking pictures and sharing photos with other people have become a fashion nowadays. Many technologies catering to these needs have burgeoned in the last decade to make the image sharing activities more efficient in terms of real-time uploading and accurate retrieval. For example, Facebook launched *f4*, a new warm Binary Large OBjects (BLOBs) storage system to achieve high storage efficiency and to provide fault tolerance [1]. SmartEye [2] selectively uploads representative images to the cloud by making use of in-network deduplication over differentiated services (DiffServ) in software-defined networks (SDN). Although these technologies successfully provide good image storage performance to a large extent, there still exists some challenges. To illustrate these challenges, let us present two typical scenarios as follows.

Scenario 1: When an earthquake or volcano eruption took place, people in the disaster area may send photos of vicinity area and other things to the Internet websites to share their current situation. Most of these pictures contain very useful information, which motivates people to search websites for pictures taken in their hometown and streets to have an understanding of their families' safety. Network congestion may happen if a large number of photos are uploaded and downloaded after the disaster, causing a long latency before desired photos can be received. If we place similar images on the same server, or servers closely connected, we can save the accessing time.

Scenario 2: Once a disaster is reported, the rescuers can schedule a more efficient rescue route according to the damage level of buildings and/or the severity of wounded people. Meanwhile, the government sectors like National Response Center manage to estimate the life and property loss of a specific area by scrutinizing photos taken from that location. If we can extract only the pictures of damaged buildings from the large pool of all photos uploaded and store similar ones taken in the same area to the same server, the sectors can get the search results faster.

Existing image storage systems, however, exhibit inefficiency when they try to handle situations in above scenarios. In these scenarios, users have a natural requirement for low latency and high accuracy when searching photos. Current works [1]–[4] cannot meet these requirements satisfactorily. First, similarities between images are not considered when storing images in the datacenters. When users search similar images, the front-end server must send requests to all data servers storing related images, which leads to a large number of concurrent data transmissions. The distributed placement of similar photos could easily cause the incast traffic congestion [5]. Since datacenters or clusters are characterized by high-bandwidth and low-latency, incast congestion may severely degrade the performance in such computing environment where the many-to-one traffic pattern is common. In the image storage system, when a large number of data servers return their query results back to the front-end server, it will result in incast network congestion with a high probability. Second, users in both scenarios want to receive photos accurately.Recent work such as SmartEye [2] cannot meet this requirement because it only uses hand-crafted features to identify similar images.

To meet the aforementioned challenges, we propose CStorage, a classification-based image storage system that achieves low latency and high accuracy. while retaining the performance of existing image uploading and storage schemes. The key in the design of our proposed CStorage system is to cluster and store similar images into the same data server. CStorage uses the state-of-the-art image feature extraction algorithm (i.e.,

deep learning) and similarity-preserving hashing functions to generate a hash code for each image. As a consequence, CStorage can retrieve images with higher accuracy. Specifically, we list the contributions of this paper as follows:

1) CStorage leverages the deep learning technique to extract image features, which are more accurate than hand-crafted image features such as PCA-SIFT.
2) CStorage significantly reduces the image retrieval latency primarily caused by incast network congestion.
3) We conduct extensive experiment with millions of images. Experimental results show the effectiveness of CStorage in reducing image retrieval latency and improving retrieval accuracy.

The paper is organized as follows. The related work of image sharing is briefly described in Section II. Section III presents the design of our proposed CStorage system. Section IV presents the performance evaluation of CStorage. Section V concludes the paper with remarks on our future work.

II. Related Work

There are several groups of existing work related to our image storage system, as will be introduced below.

A. Image Feature Extraction and Representation

Traditionally, SIFT [6] or PCA-SIFT [7] are used to extract hand-crafted features describing an image and represent them in a compact way.In recent years, deep hashing methods based on deep learning neural network [8], [9] gradually replace the hand-crafted feature extraction methods due to their prominent advantages in semantic recognition or aggregation and a large number of successful applications in image processing. Among various deep neural networks, two attract strong interest from researchers and practitioners: recurrent neural networks (RNN) [10], [11] and convolutional neural networks (CNN) [8], [12], [13]. Supervised semantics-preserving deep hashing (SSDH), for example, is one of deep hash learning algorithms that unify both classification and binary codes generation in one CNN deep learning model [14].

B. Similarity-Preserving Hashing and Aggregation

Hashing techniques have long been used to cluster data of similar properties.In this paper, we employ a specific type of hashing technique called Similarity-Preserving Hashing (SPH), or Semantic Aggregation Hashing, which is in contrast to cryptographic hash functions in that it aims to maintain resemblance between two data items by mapping similar inputs to similar hash values. Harbour [15] devised the original technique behind SPH. Locality sensitive hashing (LSH) [16]–[19] brings itself under the spotlight of the academia due to its lower probability of false positive and false negative, as well as its low computational cost. For this reason, we employ LSH in our proposed system. SmartEye [2] uses the Bloom filter [20] as input to LSH to aggregate correlated images together. Clustering hash codes can be implemented by a fast binary k-means algorithm [21].

C. Image Storage

Various distributed image storage systems in cloud have been developed [1], [2]. Facebook's warm BLOB storage system *f4* [1] groups servers into different zones and allocates images into these zones to achieve high storage efficiency.SmartEye [2] selectively uploads representative images to the cloud to save bandwidth usage in case of disasters. Existing image storage systems do not consider the similarities between images when allocating them to the data servers. In CStorage, we aim to overcome this drawback by clustering similar images and storing images from the same cluster into the same data server.

III. System Design

In this section, we present the details of the design of our proposed CStorage system.

A. Overview

Figure 1 shows an overview of the system design of CStorage. When a user uploads an image to a front-end server for storage or as a reference for image searching, this image is sent to a centralized server in the cloud.

As depicted in Figure 1, in the first step, the image hashing module extracts features from the uploaded image using a deep learning technique. The extracted features are then transformed to a hash code using a similarity-preserving hashing algorithm. In the second step, the image clustering module aggregates and clusters these hash codes. The third step is the image storage and retrieval module which is responsible for storing all user images into data servers.

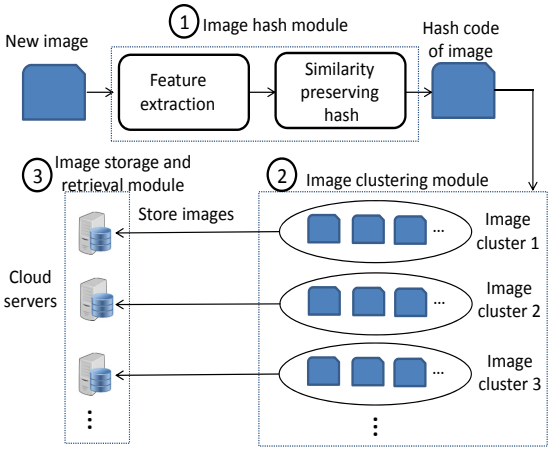

Fig. 1. Overview of the CStorage image storage system.

The image retrieval process is similar to the image uploading process. In this process, a user provides a reference image and expects to download images similar to it. The user also needs to provide the time/location information to narrow down the retrieval results.

B. Image Hashing Module

In the image hashing module, the features of an image are extracted by using a deep learning technique, and the extracted

features are then transformed to a vector of binary numbers using a similarity-preserving hashing algorithm.

After an image (or a reference image for image retrieval) is uploaded to the datacenter, the centralized server needs to extract features from this image so as to understand its semantic contents. The features extracted should be invariant to perturbations such as affine transformations and brightness variations caused by changes in camera pose and lighting. To this end, traditional methods like scale-invariant feature transform (SIFT) [6] and PCA-SIFT [7] extract hand-crafted features (i.e., keypoints) from images. However, SIFT and PCA-SIFT are proved to be ineffective and inaccurate when representing images [14]. In order to present an image with less memory usage and facilitate the image clustering process, we hash an image into a single vector of binary number. Specifically, we use the SSDH deep hashing algorithm. We adopt the 16-weight layers model of the very deep convolutional network (ConvNets) [12] to exact features from images and represent each image by a binary vector in hamming space. Comparing to the hand-crafted features, deep learning features are capable of achieving higher accuracy in semantic segmentation and object recognition [14].

We use set $Y = \{y_1, y_2, ..., y_i, ..., y_n\}$ to represent all n images in the system, where y_i is image i. After applying the ConvNets-based image hashing, each image is represent by a binary vector. That is, image y_i is represented by a d-bit binary vector $x_i = \{b_1, b_2, ..., b_d\}$, $b_d \in \{0, 1\}$. Then, the set of all images is represented by $X = \{x_1, x_2, ..., x_i, ..., x_n\}$, i.e., $X = \{X_i\}^{d \times n}$.

C. Image Clustering Module

The hundreds of millions of images stored in the datacenter need to be classified into thousands of clusters to facilitate similarity searching. The clustering algorithm is conducted periodically to update clusters of images. Once the clusters are built, when a user uploads an image, it is classified into an existing cluster. Below, we explain how to classify a large number of images to a number of clusters.

We need to consider two issues in designing an efficient clustering algorithm. First, in order to increase indexing efficiency, the similarity-preserving hash binary vectors need to be stored in the memory of the server. The SSDH deep hashing algorithm enables each image to be represented by a short hash binary vector. Thus, we can store the binary vectors of hundreds of millions of images in the memory of a single machine. Second, we need to improve the computational efficiency in clustering images. Although some clustering algorithms like kd-tree based k-means algorithm [22] can cluster a large number of images, if both the number of images and number of clusters are large, these algorithms require a long computational latency.

The Binary k-means algorithm (Bk-means) [21] was developed to overcome this challenge. Bk-means is a fast clustering algorithm on binary vectors. It builds a hash table to index all cluster centers and finds the nearest center for each image in constant time. Therefore, to perform the clustering procedure efficiently, in this paper, we adopt the Bk-means algorithm to cluster image into a set of clusters.

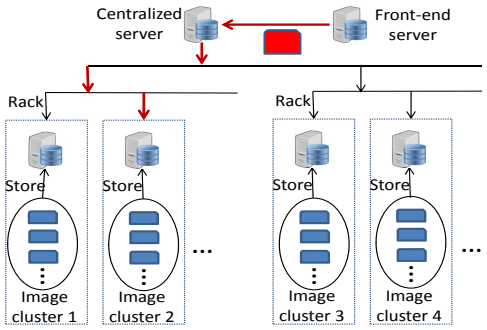

Fig. 2. Overview of the image storage and retrieval module.

Algorithm 1 Pseudocode for clustering images into clusters.

Input: image dataset $X = \{x_1, x_2, ..., x_n\}$;
initial clusters with centers $C = \{c_1, c_2, ..., c_k\}$;
Output: optimized clusters with centers $C' = \{c'_1, c'_2, ..., c'_k\}$;
while $r > 0$ //iteration not end
  for each $x_i \in X$ //assign each image to a nearest cluster
    select a cluster for x_i
  end for
  for each $c_j \in C$ //update cluster centers
    for each $c_{jq} \in c_j$ //iterate each dimension of vector c_j
      calculate $m_{jq} = \sum_{i=1}^{p} x_{iq}$
      calculate new vector c'_{jq}
    end for
  end for
  assign C' to C //use C in the next iteration
  reduce r by 1
end while
return image clusters with centers $C' = \{c'_1, c'_2, ..., c'_k\}$;

We describe the process of clustering all images in the system in Algorithm 1. The computation complexity of Algorithm 1 is $log(rnkd)$, in which r is the number of iterations, n is the number of images, k is the number of clusters and d is the number of dimensions in the image vector. Using this offline clustering algorithm, the centralized server divides all images stored in the system into a number of clusters based on the features extracted from images. To increase the clustering precision, the offline clustering algorithm is executed periodically.

D. Image Storage and Retrieval module

Figure 2 shows an overview of the image storage and retrieval module in the cloud datacenter. The centralized server is connected to all data servers in the datacenter, and data servers are divided into different racks. A particular data server is designated to store a cluster of images having similar features. These images are associated with the time/location

information, and the data server further classifies the images by them using index tables. That is, an index table records images with the same time or the same location.

Inside a datacenter, front-end servers are responsible for receiving image requests from users. There are generally two types of requests: image storage requests and image query requests. In both types of requests, a user needs to upload an image and time/location information of this image. Front-end servers then forward the requests to the centralized server.

After the centralized server receives an image from a front-end server for storage, it extracts its features and generates a hash code using the image hashing module. It then identifies its image cluster whose center is the closest to the image. It then forwards the image to the data server of the image cluster.

When a user searches for similar images, he/she uploads a reference image and specifies the time/location information of his/her desired images. This request is sent to the centralized server, which generates a hash code using the image hashing module. It then identifies the cluster that the reference image belongs to and locates the data server storing this cluster. Next it forwards the image request to the data server. The data server returns all images with the specified time/location to the front-end server. This design can be easily extended to enable users to specify time/location ranges and returning images with time/location in the specified ranges.

IV. Performance Evaluation

The contribution of the CStorage system system is its efficiency in terms of low image retrieval latency and high retrieval precision in retrieving similar images. We conduct extensive experiments and show the results in this section to demonstrate various advantages of CStorage.

A. Experiment Settings

To simulate a datacenter structure, we constructed a typical fat-tree [23] structure using 1000 data servers with 60 data servers inside each rack [24]. Each data server stores a cluster of images. We set the capacity of the downlink, uplink and buffer size of each switch to 1Gbps, 1Gbps [25] and 100kb [26], respectively. We simulated one front-end server that is connected to all the data servers in the datacenter. We generated image retrieval requests on the front-end server, and each data server receiving the request returns all similar image to the front-end server.

In our experiments, we varied the image query rates from 50 queries per second to 250 queries per second to test the performance of image storage systems [27]. When a server (either a front-end server or a data server) receives multiple image transmission requests from other servers, all incoming images are received by the server sequentially and wait in a queue. Each image is divided into a number of packets and each packet is 1kb. The sender server finishes sending an image to the receiver server when the image's last packet is received by the receiver server. The timeout of TCP packet retransmission was set to 10ms [28]. A sender server sends out a packet again if it does not receive an acknowledgement from the receiver server after 10ms, and it terminates sending this packet after 5 consecutive timeouts. If packet transmission timeout occurs when a sender server sends an image to the receiver server, we regards it as one instance of incast congestion.

We used the ILSVRC2012 image dataset, a subset of ImageNet [29] widely used in academia in the research on computer vision and image recognition [8], [21]. ILSVRC2012 contains more than 1.2 million images and 1000 clusters of similar images; each cluster roughly contains more than 1000 images. Images in a cluster are considered to be similar in the simulation and experiment. Transfer learning techniques [30] reuse previously learned image features to provide guidance for clustering. In our experiment, we adopted a publicly available CNN model [31] which is built by utilizing a transfer learning technique, so that we avoid the training process from the scratch. We manually attached time/location information to each image. Specifically, the year of each image was randomly selected in [2010, 2015] and the location of each image was randomly selected among 20 cities. An image retrieval request is generated by randomly choosing a reference image and time/location information such as "request for images that are similar to image X, taken in 2012 in city Y".

1) Comparison Methods: To evaluate the performance of CStorage in comparison with existing image storage systems, we have implemented three comparison methods including *SmartEye* [2], *f4* [1] and *Baseline*. We use the *SmartEye* approaches to create an image storage system though it is not the purpose of this work. *SmartEye* applies PCA-SIFT to represent the features of an image by a number of vectors of real numbers. In our implementation, the dimensionality of each PCA-SIFT feature was set to 36 since this value achieves the best similarity matching performance [7]. It then uses LSH to all feature vectors of an image to generate a single hash value vector for this image. It finally clusters all images based on their hash vectors and stores each image cluster into the same data server. In *f4*, each image is randomly stored in a distributed data server without considering the similarities between images. In *f4*, the centralized server uses our feature exaction and clustering techniques. It records the clustering results in a table. To retrieve similar images, it searches the table for images in the same cluster. In *Baseline*, each image is stored in a random data server without considering the similarities between images. The centralized server uses the feature exaction and clustering techniques of *SmartEye*. It records the clustering results in a table. To retrieve similar images, it searches the table for images in the same cluster.

2) Performance Metrics: We use the following five metrics to measure the performance of the image storage systems. The first two metrics represent low-latency requirement of the system while the latter two metrics measure the accuracy of retrieving similar images.

• **Average image upload latency.** This metric measures the total time used to upload an image to the designated storage server, including transmission time from the front-end server to the centralized server, time for image hashing and clustering, and transmission time of the image from the centralized server

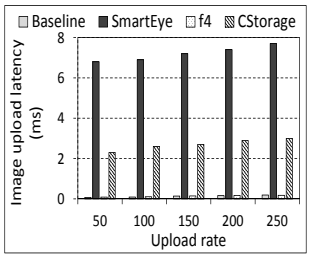

Fig. 3. Image upload latency.

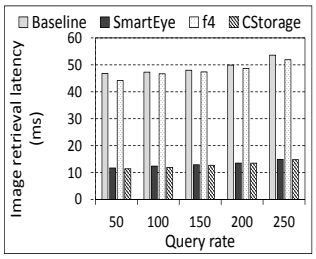

Fig. 4. Image retrieval latency.

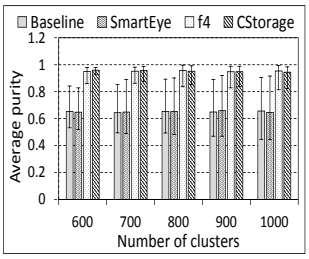

Fig. 5. Purity.

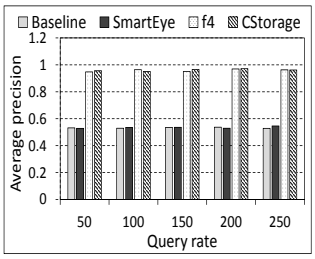

Fig. 6. Precision.

to the data server.

• **Average image retrieval latency.** This metric measures the total time used to return all similar images to the front-end server. The image retrieval latency consists of three parts: 1) upload reference image from the front-end server to the centralized server, 2) processing in the centralized server including image hashing and clustering, and 3) transmission of all similar images from data servers to the front-end server. The average retrieval latency sums up all these three parts which reflects the actual latency experienced by a user at the client end issuing a download request. This metric reflects the actual latency experienced by a user at the client end issuing a download request.

• **Purity.** Purity represents how well we cluster similar images into one cluster. For each cluster resulting from a clustering algorithm, we calculate the cluster purity of this cluster as the portion of correctly classified images in this cluster.

• **Precision.** Precision is defined by the ratio of the number of correctly returned results to the total number of returned results. It reflects the precision of the retrieved set of similar images to the reference image uploaded by the requester.

• **Recall.** Recall is defined by the ratio of the number of correctly returned results to the total number of correct results. It reflects the recall of retrieved set of similar images to the reference image uploaded by the requester.

B. Performance on Latency

Figure 3 shows the average image upload latency when the upload image rates increase from 50 images per second to 250 images per second. We see that *Baseline* and *f4* generate a shorter image upload latency due to the reason that they store an input image in a random data server without extracting its features and classifying it to an image cluster. In *CStorage* and *SmartEye*, we first need to extract features from an input image and identify the cluster that the image belongs to. We then allocate the input image into the data server of the image cluster. *SmartEye* generates a longer image upload latency than *CStorage* due to the reason that *SmartEye* uses PCA-SIFT to represent the features by a number of vectors of real numbers, and these vectors need longer latency to process when clustering the images. On the other hand, *CStorage* represents an image by features exacted from deep convolutional network, which are denoted by a single binary vector in hamming space. The binary vector needs shorter latency to process when clustering the images.

Figure 4 shows the average image retrieval latency versus image query rate. In this experiment, we varied the image query rates from 50 queries per second to 250 queries per second. From the figure, we see that the relative performance between different methods in image retrieval latency follows: *CStorage*<*SmartEye*<*f4*<*Baseline*, and *CStorage* generates a substantially shorter retrieval latency than other methods. In *Baseline* and *f4*, the images that belong to the same image cluster are randomly stored in different data servers. When the front-end server sends out an image query, the centralized server will forward the query to all data servers storing similar images. A large number of servers will transfer similar images to the front-end server in a short time. An excessive number of simultaneous responses to the front-end server will cause in-cast congestion on the front-end server side. Some packets will be dropped and the data servers need to retransmit the missing packets, leading to a long latency. *CStorage* generates a shorter image retrieval latency than *SmartEye* because *CStorage* uses features exacted from deep convolutional network to cluster images and generates shorter latency when it clusters the input image as shown in Figure 3. On the other hand, SmartEye uses PCA-SIFT to represent the features by multiple vectors of real numbers, which need longer latency to process when clustering the images compared to vectors of binary numbers used in *CStorage*.

C. Performance on Image Retrieval Accuracy

Figure 5 shows the 90^{th} percentile, median and 10^{th} percentile of the purity for different numbers of randomly selected clusters. We see that in *CStorage* and *f4*, the 90^{th} percentile purity values are close to 1, and the 10^{th} percentile purity values are higher than 0.8, which demonstrates a high clustering accuracy of these methods. This is because they uses features exacted from deep convolutional network to cluster images. Deep learning is an effective tool in clustering images and similar images are more likely to be allocated to the same data server. *SmartEye* and *Baseline* generate lower 90^{th} percentile, median and 10^{th} purity values than *CStorage* and *f4* because they use PCA-SIFT to cluster images, which are less effective in reflecting the semantic contents in the images.

Figure 6 shows the average precision versus image query rate. We see that *f4* and *CStorage* generate higher precision due to the reason that they both use features exacted from deep convolutional network to cluster images, and the returned images are likely to be allocated to the same cluster. *Baseline*

and *SmartEye* generate lower precision due to the reason they use hand-crafted features to identify similar images, which cannot effectively reflect the semantic contents in the images. Figure 7 shows the average recall with different image query rates. We see that *f4* and *CStorage* generate higher recall due to the reason that they both use features exacted from deep convolutional network to cluster images, which are effective in reflecting the semantic contents of the images. Thus, they are likely to allocate all similar images to the same cluster. *Baseline* and *SmartEye* generate lower recall due to the same reason as in Figure 6.

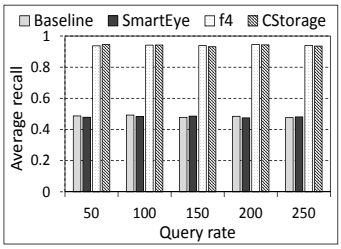

Fig. 7. Recall.

V. Conclusion and Future Work

In this paper, we presented CStorage, a novel classification-based image storage and retrieval system, which achieves low latency and high accuracy in image searching. The key of our design is to cluster similar images and store similar images in the same data server, instead of spreading randomly to all servers in the datacenter. By leveraging the state-of-the-art deep learning technique for similar image classification, CStorage manages to improve the image retrieval accuracy and recall rate. To verify the efficiency and effectiveness of CStorage, we conducted extensive experiments on a large-scale image dataset and the results shows that CStorage outperforms some image storage systems. In the future, we would make a thorough exploration of the most recent deep learning models and adapt it to CStorage to further improve the performance.

Acknowledgements

This research was supported in part by U.S. NSF grants OAC-1724845, ACI-1719397 and CNS-1733596, and Microsoft Research Faculty Fellowship 8300751. We would like to thank Dr. Yuhua Lin for his help in this work.

References

[1] Subramanian Muralidhar, Wyatt Lloyd, Sabyasachi Roy, Cory Hill, Ernest Lin, Weiwen Liu, Satadru Pan, Shiva Shankar, Viswanath Sivakumar, Linpeng Tang, and Sanjeev Kumar. f4: Facebook's warm blob storage system. In *Proc. of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2014.

[2] Yu Hua, Wenbo He, Xue Liu, and Dan Feng. Smarteye: Real-time and efficient cloud image sharing for disaster environments. In *Proc. of the IEEE Conference on Computer Communications (INFOCOM)*, 2015.

[3] Doug Beaver, Sanjeev Kumar, Harry C Li, Jason Sobel, and Peter Vajgel. Finding a needle in haystack: Facebook's photo storage. In *Proc. the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2010.

[4] H. Wang, H. Shen, and G. Liu. Swarm-based incast congestion control in datacenters serving web applications. In *Proc. of SPAA*, 2017.

[5] Haitao Wu, Zhenqian Feng, Chuanxiong Guo, and Yongguang Zhang. Ictcp: Incast congestion control for tcp in data-center networks. *IEEE/ACM Transactions on Networking*, 21(2):345 – 358, 2013.

[6] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[7] Y. Ke and R. Sukthankar. Pca-sift: A more distinctive representation for local image descriptors. In *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2004.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proc. of the Annual Conference on Neural Information Processing Systems (NIPS)*, 2012.

[9] Jianlong Fu, Tao Mei, Kuiyuan Yang, Hanqing Lu, and Yong Ru. Tagging personal photos with transfer deep learning. In *Proc. of the International World Wide Web Conference (WWW)*, 2015.

[10] Wim De Mulder, Steven Bethard, and Marie-Francine Moens. A survey on the application of recurrent neural networks to statistical language modeling. *Computer Speech & Language*, 30(1):61–98, Mar. 2015.

[11] A. Graves, M. Liwicki, S. Fernandez, R. Bertolami, H. Bunke, and J. Schmidhuber. A novel connectionist system for improved unconstrained handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 31(5):855 – 868, 2009.

[12] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[13] H. Sak, A. W. Senior, and F. Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *Proc. of the Annual Conference of the International Speech Communication Association (Interspeech)*, 2014.

[14] Huei-Fang Yang, Kevin Lin, and Chu-Song Chen. Supervised learning of semantics-preserving hashing via deep neural networks for large-scale image search. *arXiv preprint arXiv:1507.00101*, 2015.

[15] N. Harbour. Dcfldd. defense computer forensics lab. In *Available: http://dcfldd.sourceforge.net*. 2002.

[16] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proc. of ACM Symposium on Theory of Computing (STOC)*, 1998.

[17] J. Tang, Z. Li, L. Zhang, and Q. Huang. Semantic-aware hashing for social image retrieval. In *Proc. of the Annual ACM International Conference on Multimedia Retrieval (ICMR)*, 2015.

[18] F. Zhao, Y. Huang, L. Wang, and T. Tan. Deep semantic ranking based hashing for multi-label image retrieval. *arXiv preprint arXiv:1501.06272*, 2015.

[19] L. Li, C. C. Yan, W. Ji, B.-W. Chen, S. Jiang, and Q. Huang. Lsh-based semantic dictionary learning for large scale image understanding. *Journal of Visual Communication and Image Representation*, 31:231–236, 2015.

[20] B. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[21] Yunchao Gong, Marcin Pawlowski, Fei Yang, Louis Brandy, Lubomir Boundev, and Rob Fergus. Web scale photo hash clustering on a single machine. In *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[22] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisser-man. Object retrieval with large vocabularies and fast spatial matching. In *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2007.

[23] G. Liu, H. Shen, and H. Wang. Computing load aware and long-view load balancing for cluster storage systems. In *Proc. of Big Data*, 2015.

[24] T. Benson, A. Akella, and D. A. Maltz. Network Traffic Characteristics of Data Centers in the Wild. In *Proc. of Internet Measurement Conference (IMC)*, 2010.

[25] Cisco Nexus 3064 Switch. http://www.cisco.com/c/en/us/products/switches/nexus-3064-switch/.

[26] S. Shukla, S. Chan, A. Tam, A. Gupta, Y. Xu, and H. Chao. Tcp plato: packet labelling to alleviate time-out. *IEEE Journal on Selected Areas in Communications*, 32(1):65–76, 2014.

[27] G. Buehrer, B. W Weide, and P. A. Sivilotti. Using parse tree validation to prevent sql injection attacks. In *Proc. of the International workshop on Software engineering and middleware (SEM)*, 2005.

[28] V. Paxson, M. Allman, J. Chu, and M. Sargent. Computing TCP's Retransmission Timer. Technical report, RFC2988, November, 2011.

[29] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[30] A. Quattoni, M. Collins, and T. Darrell. Transfer learning for image classification with sparse prototype representations. In *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008.

[31] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 22(10):1345–1359, 2010.