Content Sifting Storage: Achieving Fast Read for Large-scale Image Dataset Analysis

Yu Liu
Huazhong University of Science and Technology
Wuhan, China
liu_yu@hust.edu.cn

Hong Jiang
University of Texas at Arlington
Arlington, TX US
hong.jiang@uta.edu

Yangtao Wang
Huazhong University of Science and Technology
Wuhan, China
ytwbruce@hust.edu.cn

Ke Zhou✉
Huazhong University of Science and Technology
Wuhan, China
k.zhou@hust.edu.cn

Yifei Liu
Stony Brook University
Stony Brook, NY US
yifeiliu@cs.stonybrook.edu

Li Liu
Huazhong University of Science and Technology
Wuhan, China
lillian_hust@hust.edu.cn

***Abstract*—Analyzing large-scale image dataset requires all images to be read from disks first, leading to high read latency. Therefore, we propose a Content Sifting Storage (CSS) system, which aims to reduce the read latency by only reading sifted relevant data. CSS generates embedded content metadata via deep learning and manages the metadata via Semantic Hamming Graph, which achieves fast read based on content similarity meeting the given analysis. Extensive experimental results on image datasets show that compared with conventional semantic storage systems, our CSS can greatly reduce the read latency by 82.21% to 94.8% with more than 98% recall rate.**

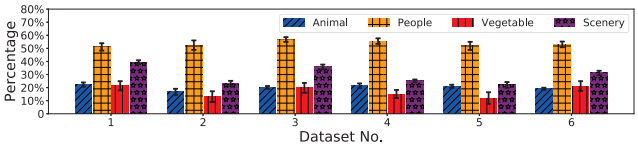
***Index Terms*—Content Sifting Storage, Semantic Hamming Graph, read latency, large-scale image dataset**

I. Introduction

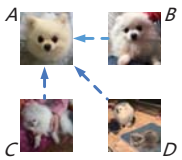
Images are frequently used for data analysis because of their rich semantic content. Due to the relatively large size of image files, the latency of reading all images before image processing has gradually become a performance bottleneck, especially in the increasing volumes of data. Take 100 million images of size 1MB each for example, with the average sequential read bandwidth of SATA hard disk on the servers of company $Tencent$ (one of the largest social network companies in the world) being around 220MB/s, it will take about 5.26 days to read these images. While distributed storage offers an initial solution to this problem, the increasing amount of data in image datasets and the scaling out of distributed storage make this a non-trivial problem. Therefore, it is our belief that a more fundamental solution is called for that addresses this problem on a single node, which in turn will complement and enhance a distributed solution.

In practice, we find that not all images read from disks will be actually used for the given analysis. As shown in Fig. 1(a), for the analyses of image content including animal, people, vegetable and scenery, the percentages of the relevant data in different datasets are averagely 20.11%, 53.45%, 17.01% and 29.43%. Subsequently and generally, large amounts of data are put aside after simple processing that determined their irrelevance to the analysis, but after they have already incurred huge read latency. Based on this observation, we decide whether it is beneficial to only read relevant data to save bandwidth, where the set of relevant data is determined by content sifting. Content sifting is a process that first measures the semantic relevance between data content and the analysis, and then indexes the relevant data. That is, if the sum of the latencies incurred by sifting and reading the sifted dataset is likely to be less than the read latency of the original dataset, then sifting becomes beneficial since, the more the unrelated data, the more bandwidth will be saved. Based on the metadata which expresses content of images, a concrete and integrated read process is to first embed the requirement of analysis into the metadata, then sift and index relevant data by content similarity, and finally read the sifted data. However, there are two key challenges in defining the proper metadata for data sifting and reading for unstructured image data analysis.

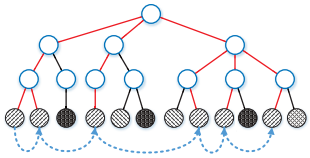
✉Corresponding author: Ke Zhou.



(a) Percentages of 6 image datasets are actually used for 4 analyses. These datasets are collected from servers that store album data in company $Tencent$.



(b) Relevant content for analysis of A.



(c) Directory structure is hard to adopt similarity to achieve fast reads.

Fig. 1. Motivation of Content Sifting Storage.

First, conventional metadata cannot meet the expression of semantic relevance for content sifting demands. Conventional metadata that expresses unstructured data content by text labels has several drawbacks, including high cost of hand-crafted annotation, lack of uniform description, time-consuming comparison, *etc.*, rendering them unsuitable and impractical. On the other hand, hash codes have been widely used as metadata because of their compact binary codes and fast exclusive-OR (XOR) comparisons that make them lightweight and efficient. Using Locality Sensitive Hash (LSH) [1] in semantic storage can not only embed data content without relying on hand-crafted labels, but also achieve efficient comparison. Nevertheless, LSH also has limitations. As shown in Fig. 1(b), while LSH can estimate content similarity according to the global pixel distribution of an image (*i.e.*, A is similar to B), it fails to understand semantic relevance on different postures (*i.e.*, A and C) and local content similarity (*i.e.*, A and D). However, these semantically similar images (*i.e.*, C and D) are equally important for the analysis of A.

978-1-7281-1085-1/20/$31.00 ©2020 IEEE

Second, conventional metadata structures lack efficient mechanisms for reading data based on content similarity. Conventional storage systems [2], [3] by and large manage data by a directory structure that identifies data by its name-path, mostly ignoring the content relationships among data items, and making it hard to take full advantage of similarity to achieve fast reads. As shown in Fig. 1(c), although similar data items are linked by the path on the blue dotted line that hints at sequential reads of these similar data, the directory structure does not allow for such sequential reads. While the graph-based structure is considered a flexible way to manage relationships [4], the content similarity between data items in existing methods, defined by whether there is an edge between them (represented by nodes), is too simple to express the degree of relevance. FishStore [5] records and puts the location of new data into the subspace by customized text labels, then searches matched data in the subspace for the given requirement, thus reducing the read latency. However, this scheme has a high probability to degenerate to the traditional read mechanism when missing customized labels.

To overcome the above challenges, we propose a Content Sifting Storage (CSS) system, which aims to sift and only read those data that are relevant to the analysis to reduce read latency for large-scale image dataset analysis. We adopt deep similarity hash codes as embedded content metadata to meet the expression of semantic relevance and design Semantic Hamming Graph (SHG) to achieve fast read through content similarity. On the one hand, we use the Deep Self-taught Hashing (DSTH) [6], [7] algorithm to map embedded content metadata. DSTH combines the advantages of unsupervised hashing and deep hashing, which owns generalization ability to map the unseen data. Moreover, DSTH expresses valid content of an image and owns the ability to measure relevance by Hamming distance, which is more precise than LSH in expression of semantic relevance. On the other hand, we use the metadata to build SHG which takes the hash code as vertex and expresses the relevance degree (*i.e.*, Hamming distance) between data on the edge. Therefore, CSS can index the relevant data through graph traversal to achieve fast read in the precise similarity range. Specifically, we implement CSS based on the OpenStack Swift[1] system and Neo4j [8] database, where Swift is used to store raw images and Neo4j is a graph database used to store embedded content metadata as well as build and manage SHG. We design a variety of analyses for testing the recall rate and read latency of CSS on large-scale public and real-world datasets. Extensive experimental results show that compared with conventional semantic storage systems, our CSS is able to substantially reduce the read latency, by 82.21% to 94.8%, with more than 98% recall rate.

II. CSS Methodology

A. Read Latency

We assume that the total number of images in a dataset is $\mathcal{C}$, the number of relevant data for the given analysis $\mathcal{A}$ is $\mathcal{P}$, the average size of an image is $\mathcal{I}$B, the read bandwidth of disk is $\mathcal{V}$B/s, and the latency for each image in content sifting is $\mathcal{J}$s. Let $\alpha(\mathcal{A})$ denote the latency of reading all data, and $\beta(\mathcal{A})$ the total latency of sifting all data (content) and reading relevant data, where

$$\alpha(\mathcal{A}) = \frac{\mathcal{C} \times \mathcal{I}}{\mathcal{V}}$$

$$\beta(\mathcal{A}) = \mathcal{C} \times \mathcal{J} + \frac{\mathcal{P} \times \mathcal{I}}{\mathcal{V}}$$

[1]https://github.com/openstack/swift.

We denote the difference between above latencies as

$$\Omega(\mathcal{A}) = \alpha(\mathcal{A}) - \beta(\mathcal{A}) = \mathcal{C}[\frac{\mathcal{I} \times (1 - \mathcal{P}/\mathcal{C})}{\mathcal{V}} - \mathcal{J}]$$

Obviously, the bigger the Ω is, the more efficient our proposed method will be. In the real world, the pixel level of cameras equipped on mobile devices has been increasing rapidly. As a result, the growth rate of $\mathcal{I}$ will certainly exceed that of $\mathcal{V}$ (*i.e.*, $\mathcal{I}/\mathcal{V} > 1$) with time. Meanwhile, the data resources will be increasingly abundant and what applications focus on will frequently change, which means that the proportion of required resources for a given analysis in total resources, *i.e.*$\mathcal{P}/\mathcal{C}$, will be getting smaller, so $1 - \mathcal{P}/\mathcal{C}$ will become larger. In addition, using hash code to sift data has nearly approached the limit of comparative cost, which leads to a basically stable $\mathcal{J}$. In summary, Ω will grow larger with time.

B. Deep Similarity Hash

Compared with binary codes generated by LSH, similarity hash codes can precisely measure the semantic relevance between data content by Hamming distance. This kind of deep similarity hash method no longer maps data by global pixel distribution, but relies on convolution neural network (CNN) [6] to detect and map the valid region of each image, so as to provide more relevant data for data analysis [9].

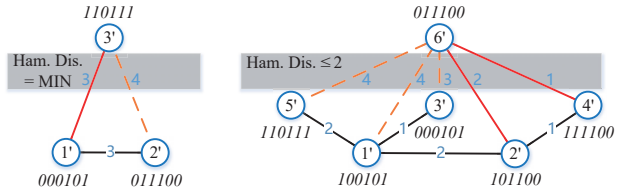
In practice, we adopt DSTH consisting of two steps to obtain the deep similarity hash code of each image. In the first step, DSTH constructs deep features extracted from pre-trained model into a graph and utilizes graph embedding technique to map these features into hash codes. In doing so, the generated hash codes are able to perceive the key information of an image by the generalization ability. In order to improve the on-line efficiency of hash mapping, in the second step, DSTH learns the above generated hash codes as labels to achieve an end-to-end model for real-time feature extraction and hash code generation.

DSTH provides the Hamming distance which is acquired by executing XOR on a pair of hash codes, to express the semantic relevance between them. Shorter Hamming distance implies that hash codes are more semantically relevant (containing the same or similar objects). The most similar images share the same hash code, thus the Hamming distance between them is 0. In practice, DSTH achieves the best precision-recall performance on the 48-bit code length, and the best retrieval precision when the Hamming radius is no more than 2 [10].

C. Semantic Hamming Graph

The proposed semantic Hamming graph methodology aims to organize embedded content metadata (*i.e.*, similarity hash codes) efficiently. For N images in a storage system, each hash code corresponding to an image is represented as a node in graph G. Considering some images share the same hash code, we merge the same hash codes into one node and $M(\leq N, \in Z^+)$ denotes the number of different hash codes as well as the number of nodes [11]. Meanwhile, the Hamming distance between two hash codes stands for the weight of an edge that links each two nodes. The edge is undirected because the relevance is mutual.

As for G containing M nodes, the maximal number of edges is $M \times (M-1)/2$ in the case of fully connected nodes. It will certainly cause huge latency and burden for processing such a graph that contains too many links. Therefore, we set a threshold T for restricting the number of edges. Let V_* denote $*$-th node of G and $H(V_*)$ is the hash code of V_*. For any i and j $(0 < i, j \leq M, i \neq j)$, the Hamming distance $\mathcal{F}_{ij}$ between two hash codes V_i and V_j can



(a) example for $T =$ minimal Hamming distance

(b) example for $T \leq 2$

Fig. 2. The choice of threshold T.

be calculated by $H(V_i) \oplus H(V_j)$ where $\oplus$ is XOR operation. We denote $\mathcal{F}_{ij}$ as

$$\mathcal{F}_{ij} = \mho(H(V_i), H(V_j))$$

Let T_i denote the Hamming distance threshold of the i-th node. For $\forall i \in (0, M]$, the choice of T_i can be interpreted as:

$$T_i = \begin{cases} min\mathcal{F}_{ij} & j \neq i, j \in (0, M], if \quad min\mathcal{F}_{ij} > 2 \\ 2 & \text{otherwise} \end{cases}$$

In order to decrease the number of edges, we stipulate there exists an edge with V_i, only if

$$\mathcal{F}_{ij} \leq T_i$$

As shown in Fig. 2(a), node 3 is a newcomer. It computes the Hamming distance from all existing nodes, and the minimum distance is more than 2. In this case, node 3 is only connected with node 1 because they have the shortest Hamming distance. As shown in Fig. 2(b), node 6 is a newcomer. In this case, node 6 is connected with node 2 and node 4 because their Hamming distances are 2 and 1 respectively.

The definition of T_i is based on two principles. One is that the minimal Hamming distance can ensure that there are no isolated nodes in Graph G as well as effective data query within predefined Hamming radius. It is worth mentioning that the shortest edges can ensure at least a recall result from retrieval nodes within the Hamming radius. For any i and j $(0 < i, j \leq M, i \neq j)$, assuming that the indirect Hamming distance between two hash codes is $U(H(V_i), H(V_j)$, where there is at least one node V_k $(k \neq i, k \neq j, k \in (0, M])$ in G satisfying $U(H(V_i), H(V_j) = \mho(H(V_i), H(V_k)) + \mho(H(V_k), H(V_j))$, the following must hold:

$$min\{U(H(V_i), H(V_j)\} \geq \mho(H(V_i), H(V_j))$$

The other is that the accuracy achieves the best performance under the same recall rate when the Hamming radius is no more than 2, according to Section II-B. Therefore, the value of T_i we set can promote the query efficiency. When querying on SHG for an analysis, the time complexity is linear and expressed as $O(|M_t| + |E_t|)$ where M_t is the number of used nodes and E_t denotes the number of edges during traversal.

III. CSS Design

A. CSS overview

Fig. 3 shows the architecture and workflow of CSS. CSS is mainly composed of three parts: DSTH model, SHG and file storage. DSTH model generates similarity hash code as embedded content metadata for each image. This embedding process occurs outside the critical path of user-oriented operations after the data are stored. To obtain more precise embedded content metadata, CSS configures 48-bits hash codes according to Section II-B. Subsequently, as the most crucial component of CSS, SHG leverages Hamming distance and graph structure to organize the embedded content metadata generated by DSTH model. Note that it is a new independent metadata management scheme without affecting native metadata organizations, attempting to provide accurate and fast content-based queries. SHG can speed up the sifting process by clustering semantically relevant objects into adjacent areas of the graph. In addition, this graph model can be updated incrementally since the relationships between former hash codes are certain and only new hash codes should be added to it. In the bottom of CSS, file storage reads a file by its name-path.

We design CSS on the top of OpenStack Swift storage system. It is worth mentioning that the white dotted frame part of CSS works as a system middleware, which can be applied to any existing file storage systems and is an independent component working with searchable metadata systems such as Sypglass [12], SmartStore [2], and FAST [1].

When a new image comes, CSS will generate its embedded content metadata by DSTH model and store original data into file storage system by traditional way. Then the hash code along with file name is sent to SHG, and SHG estimates whether this hash code already exists. If it exists, SHG will edit the property of node corresponding to the hash code by adding this file name. Otherwise, SHG will add a new node, and add this hash code along with its file name to the property of this node. Furthermore, SHG will generate new connections according to the principle in Section II-C.

When an analysis requirement comes, this requirement will be represented by one or more images with expected sifting radius and sent to DSTH model. DSTH generates the embedded content (similarity hash codes) of these images, and sends them along with corresponding radius to SHG. SHG retrieves a set of metadata according to the hash code and its radius, organizes the file names from the set of metadata into a subset, and then sends this subset to file storage layer. According to these file names, file storage system reads and returns the corresponding files to the memory for data analysis.

B. SHG management

Compared with a tree structure, SHG plays a key role in speeding up reading of metadata base on similarity shown in Fig. 4. As metadata structure of CSS, it not only records correlations of metadata but

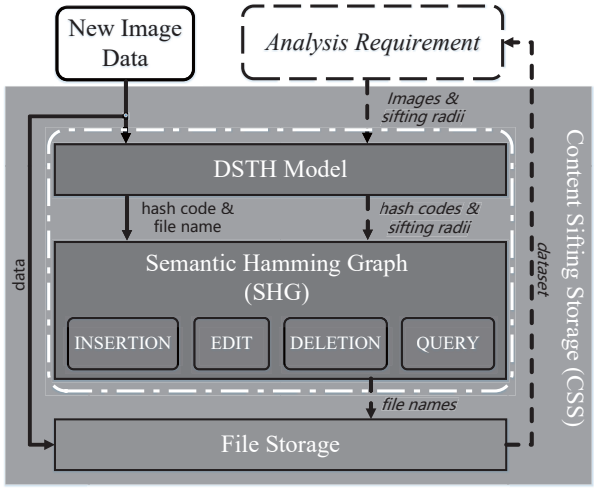


Fig. 3. CSS architecture.

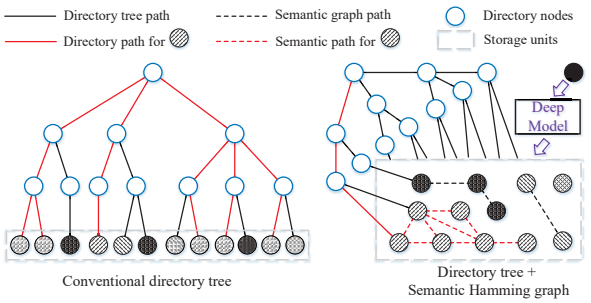


Fig. 4. Comparison with conventional directory structure in term of query.

also provides the interfaces for content sifting. The implementation of SHG in our prototype relies on a graph database Neo4j. By taking advantage of the functionality of Neo4j, SHG can sustain large amounts of data and use specialized query interfaces. CSS applies these specialties to simplify and accelerate content sifting processes. In practice, each node has four properties, *i.e.*, *hash code*, *file name*, *edge* and *note*.

It is worth noting that the *hash code* denotes the node's ID in the graph. If there exists hash collision that multiple file names share one hash code, we use "&" as symbol to concatenate file names in the property of *file name*. The *edge* records the connection relationships with their weights of the node. The *note* is an emergency property. *Edge* and *note* are automatically maintained by the graph database system. In addition, we provide four interfaces (*i.e.*, INSERTION, EDIT, DELETION and QUERY) for **node operation on the graph**.

INSERTION: For the newly appeared hash code, we create a new node in the graph database. The property of *hash code* records new hash code. The property of *file name* records new file name. The property of *edge* will be generated by the connection principle in Section II-C.

EDIT: If the file name corresponding to existing hash code changes, the edit function will be triggered. It may result from the change of file name, the addition of file or the deletion of file.

DELETION: When the last file name in the node is deleted which leads to an empty *file name*, the node corresponding to this hash code along with its connections will be removed from SHG.

QUERY: Query on SHG needs two parameters: query node (*i.e.*, a hash code) and sifting radius R ($R \in N$). According to Breadth First Search (BFS), from the query node, SHG explores the nodes by weights on edges and collects nodes within the sifting radius into the subset. Finally, SHG returns the *file name* of all nodes in the subset.

IV. Performance Evaluation

A. Dataset

To test the performance of CSS, we conduct experiments on three image datasets, *i.e.*, ImageNet [13], MS-COCO [14] and a real-world dataset. We summarize the characteristics of these datasets below.

ImageNet: ImageNet is a public image database which contains over 14 million URLs of images. The used images in the experiment are from classification-localization dataset which includes a total of 1,281,167 images for training. We use all the 1,281,167 images to test our prototype in the big data circumstance. ImageNet can be used for content sifting test of global target.

MS-COCO: MS-COCO is a public dataset for image recognition, segmentation and captioning. The current release contains 118,287 training images, 40,504 validation images and 40,775 test images, where each image is labeled by some of the 80 semantic concepts. We use all the 199,566 images to test our prototype in the big data circumstance. MS-COCO can be used for content sifting test of partial target.

Real-world dataset: This dataset is collected from company $Tencent$ during a certain period, whose size is around 5TB consisting of 1,000,000 images. It can be used for content sifting test in real world.

B. Experimental Setup

We have implemented the CSS prototype in Ubuntu 16.04 with OpenStack Swift Storage 2.16.1. The performance evaluation is conducted on two servers, with a 2-node distributed OpenStack Swift storage system, each of which has two 10-core Intel Xeon E5-2640 CPU, 64GB DDR4 memory, and a single NVIDIA Tesla K40m GPU. The hash codes generation uses TensorFlow [17] which provides stable Python APIs. Considering OpenStack Swift is written in Python, for compatibility, we implement the CSS prototype by Python.

Recall rate and read latency are two most important performance metrics in this study. As for recall rate measurement, we can take advantage of the labels of public datasets (ImageNet and MS-COCO) to test. The read latency can be tested on real-world dataset with 8TB SATA hard disk whose sequential read bandwidth is around 500MB/s. For ImageNet, we choose *house*, *fish* and *bird* as the analysis requirement. Similarly, *person*, *car* and *sky* are chosen for MS-COCO while *person*, *scenery* and *street* are chosen for real-world dataset. These objects account for a relatively large proportion in their respective dataset, which is conducive to data display.

The comparisons of CSS with other metadata search systems are manifested in two aspects: one is using embedded content metadata to improve recall rate and the other is using weighted graph structure to accelerate query processes. We list the differences in Table I. Note that both SmartStore and Spyglass use text labels for full scan, so they will not participate in the recall rate comparison. In addition, each image in public datasets almost occupies the same storage space, so the read latency is basically the same for each one. Therefore, the latency we compare on public datasets represents the sifting latency in metadata. Moreover, in order to better show this latency comparison, we display the cumulative latency of executing 100,000 queries. On real-world dataset, we manifest a total read latency [18] comparison where the outset is that the client posts request and the termination is that all data have been read to memory. It should be noted that all data have already been stored in respective system before carrying out the experiments. In particular, R denotes sifting radius.

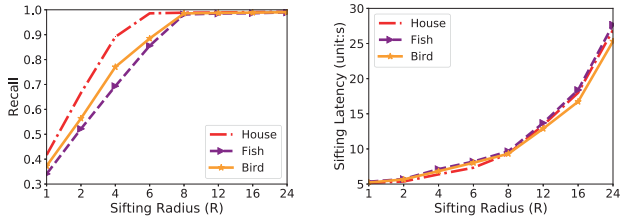
C. Performance Comparison Varying R

When using hash codes as metadata, some relevant data could not be found because of the inherent accuracy loss of hash embedding. Therefore, we focus the recall rate primely. In CSS, the recall rate relies on both the quality of hash code and the value of R. The increase of R will promote the recall rate but also increase the sifting latency. Therefore, we list the changes of recall rate and sifting latency when varying R on public datasets. It should be noted that for the same analysis requirement, we use 10 different images as query points to independently test with each R, so each key point on the curve shows the average result after 10 tests.

As shown in Fig. 5(a), both the recall rate and sifting latency increase with an increasing R on ImageNet, and the recall rate reaches 98.12% at least on *fish* analysis when $R = 8$. Furthermore, the recall rate increases slowly after $R = 8$. However, as shown in Fig. 5(b),

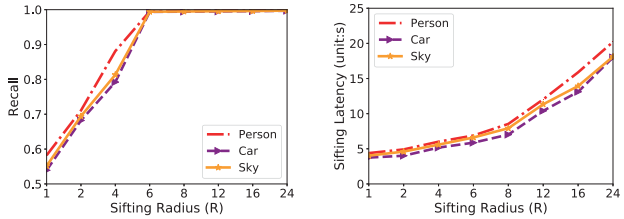
TABLE I
COMPARISON OF CSS WITH OTHER METADATA SEARCH SYSTEMS

	CSS	FAST [1]	SmartStore [2]	Spyglass [12]
Feature Extraction	Convolutional Neural Network [6]	PCA-SIFT [15]	Traditional Metadata	Traditional Metadata
Metadata Arrangement	DSTH [7]	Locality Sensitive Hashing [16]	Latent Semantic Indexing	Hierarchical Partition
Organizational Structure	SHG	Cuckoo Hashing	R-Tree	K-D Tree



(a) Recall rate with different R (b) Sifting latency with different R

Fig. 5. Performances with different R on ImageNet.



(a) Recall rate with different R (b) Sifting latency with different R

Fig. 6. Performances with different R on MS-COCO.

the sifting latency continues to rise fast after $R = 8$. As a result, we achieve the best outcome when $R = 8$ on ImageNet.

Similarly, as shown in Fig. 6, the best benefit can be captured when $R = 6$ on MS-COCO, and the recall rate reaches 99.37% at least. Different from ImageNet, the best result of MS-COCO comes from a smaller R, which may result from that DSTH is able to achieve better results when dealing with images containing mixed content.

D. Performance Comparison with Semantic Storage Systems

In this part, we compare the sifting latency to verify that our CSS accelerates the data reading because of our well-designed data structure and sifting way. In addition, compared with FAST, we list the recall rate of relevant data to verify that using deep similarity hash code as metadata is more suitable for sifting data in analysis scenes. Similarly, we also use 10 different images to independently test for the same analysis requirement. According to section IV-C, we choose $R = 8$ and $R = 6$ to carry out this experiment on ImageNet and MS-COCO respectively. We use K-D Tree and R-Tree to simulate Spyglass and SmartStore respectively.

Table II lists the recall rate and sifting latency of three analysis requirements (*i.e.*, house, fish and bird) respectively on ImageNet. In terms of recall rate, compared with FAST, CSS is superior and averagely reaches 98.5%, which shows deep similarity hash is more suitable than LSH for data sifting in analysis scenes. As for sifting latency, CSS produced the least latency in all the three analysis requirements, because CSS has improved both data structure and sifting way, while other three systems have been improved only in one of these two aspects. In addition, although the latency of FAST is close to ours, it missed too much relevant data, thus making it meaningless to compare CSS with FAST. Compared with Spyglass and SmartStore, CSS shortened the shortest latency by nearly 600 times.

TABLE II
RECALL AND SIFTING LATENCY COMPARISON ON IMAGENET.

	Systems	Analysis Requirements		
		house	fish	bird
recall	FAST	0.3011±0.0288	0.2612±0.0193	0.2779±0.0344
	CSS	0.9883±0.0003	0.9812±0.0016	0.9855±0.0006
sifting latency (unit:s)	Spyglass	6.5128×10^3	6.5128×10^3	6.5128×10^3
	SmartStore	7.6917×10^4	7.6917×10^4	7.6917×10^4
	FAST	32.3232±0.0003	32.3231±0.0002	32.3232±0.0002
	CSS	9.3190±0.0021	9.6606±0.0018	9.3102±0.0020

TABLE III
RECALL AND SIFTING LATENCY COMPARISON ON MS-COCO.

	Systems	Analysis Requirements		
		person	car	sky
recall	FAST	0.2333±0.0144	0.1537± 0.0126	0.1667±0.0131
	CSS	0.9950±0.0011	0.9937±0.0008	0.9942±0.0012
sifting latency (unit:s)	Spyglass	8.7212×10^2	8.7212×10^2	8.7212×10^2
	SmartStore	3.1687×10^3	3.1687×10^3	3.1687×10^3
	FAST	6.1873±0.0002	6.1871±0.0002	6.1871±0.0001
	CSS	6.8902±0.0031	5.8701±0.0023	6.6084±0.0025

Table III lists the recall rate and sifting latency of three analysis requirements (*i.e.*, person, car and sky) respectively on MS-COCO. The recall rate of CSS outperforms that of FAST again, and averagely reaches 99.43%. In terms of sifting latency, CSS has averagely shortened the latency by 100 times compared with Spyglass.

Above comparisons show that CSS can greatly decrease the data sifting latency on the premise of high recall rate of relevant data, as well as has the potential to be applied to large-scale real-world image datasets.

E. Latency Comparison on Real-World Dataset

Finally, we validate our system on the real-world dataset. The starting point is that the client posts request, and the end point is that all data have been read to memory. We collect the latency during this period (we ignore the latency of data replacement in memory).

Table IV lists the data read latency responding to the analysis requirements on real-world dataset. We can see that the read latency on real-world dataset presents our overwhelming advantage and the same trend as that on public datasets with the change of R. Besides, the longest read latency of CSS is only 17.79%, 7.96%, and 5.20% of the shortest read latency of other systems, respectively. This may reflect the proportion of relevant data in the total data, but also verify our efficiency and the ratiocination in section II-A that more unrelated data will lead to less read latency.

V. CONCLUSION

In this paper, we introduce deep similarity hash codes and weighted graph structure into storage systems, aiming at alleviating the latency

TABLE IV
LATENCY COMPARISON ON REAL-WORLD DATASET.

Systems	Analysis Requirements		
	person	scenery	street
Spyglass	2.1121h	2.1121h	2.1121h
SmartStore	2.2271h	2.2271h	2.2271h
CSS (R=2)	0.0918h	0.0452h	0.0384h
CSS (R=6)	0.2816h	0.1158h	0.0889h
CSS (R=8)	0.3757h	0.1681h	0.1098h

caused by reading all data from the disks for large-scale image dataset analysis. We design CSS with embedded content metadata and SHG, which sifts the data meeting the analysis and indexes them to achieve fast reads. Extensive experiments on public and real-world datasets demonstrate the excellent effects of method compared with conventional semantic storage systems that CSS can greatly reduce the read latency by 82.21% to 94.8% with more than 98% recall rate.

ACKNOWLEDGMENT

This work is supported by the National Natural Science Foundation of China No.61902135 and the Innovation Group Project of the National Natural Science Foundation of China No.61821003. Wuhan is a great city whose residents are all heroes. We will surely overcome the COVID-19 and harvest freedom in the end.

REFERENCES

[1] Y. Hua, H. Jiang, and D. Feng, "Fast: Near real-time searchable data analytics for the cloud," in *SC*, 2014.

[2] Y. Hua, H. Jiang, Y. Zhu, D. Feng, and L. Tian, "Smartstore: A new metadata organization paradigm with semantic-awareness for next-generation file systems," in *SC*, 2009.

[3] J. Zhang, Y. Liu, K. Zhou, G. Li, Z. Xiao, B. Cheng, J. Xing, Y. Wang, T. Cheng, L. Liu, M. Ran, and Z. Li, "An end-to-end automatic cloud database tuning system using deep reinforcement learning," in *SIGMOD*, 2019, pp. 415–432.

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012.

[4] N. Hochgeschwender, S. Schneider, H. Voos, H. Bruyninckx, and G. K. Kraetzschmar, "Graph-based software knowledge: Storage and semantic querying of domain models for run-time adaptation," in *2016 IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*, Dec 2016, pp. 83–90.

[5] D. Xie, B. Chandramouli, Y. Li, and D. Kossmann, "Fishstore: Faster ingestion with subset hashing," in *SIGMOD*, 2019, pp. 1711–1728.

[6] K. Zhou, Y. Liu, J. Song, L. Yan, F. Zou, and F. Shen, "Deep self-taught hashing for image retrieval," in *MM*, 2015, pp. 1215–1218.

[7] Y. Liu, J. Song, K. Zhou, L. Yan, L. Liu, F. Zou, and L. Shao, "Deep self-taught hashing for image retrieval," *IEEE Trans. Cybernetics*, vol. 49, no. 6, pp. 2229–2241, 2019.

[8] J. Webber, "A programmatic introduction to neo4j," in *SPLASH*, 2012.

[9] H. Wang, X. Yi, P. Huang, B. Cheng, and K. Zhou, "Efficient SSD caching by avoiding unnecessary writes using machine learning," in *ICPP*, 2018, pp. 82:1–82:10.

[10] Y. Liu, Y. Wang, K. Zhou, Y. Yang, and Y. Liu, "Semantic-aware data quality assessment for image big data," *Future Gener. Comput. Syst.*, vol. 102, pp. 53–65, 2020.

[11] Y. Wang, Y. Liu, Y. Liu, K. Zhou, Y. Yang, J. Zeng, X. Xu, and Z. Xiao, "Analysis and management to hash-based graph and rank," in *Web and Big Data - Third International Joint Conference, APWeb-WAIM 2019, Chengdu, China, August 1-3, 2019, Proceedings, Part I*, 2019, pp. 289–296.

[12] A. W. Leung, M. Shao, T. Bisson, S. Pasupathy, and E. L. Miller, "Spyglass: Fast, scalable metadata search for large-scale storage systems." in *FAST*, 2009.

[14] T. Lin, M. Maire, S. J. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: common objects in context," in *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V*, 2014, pp. 740–755.

[15] Y. Ke and R. Sukthankar, "Pca-sift: A more distinctive representation for local image descriptors," in *CVPR*, 2004.

[16] S. Har-Peled, P. Indyk, and R. Motwani, "Approximate nearest neighbor: Towards removing the curse of dimensionality," *Theory of computing*, vol. 8, no. 1, pp. 321–350, 2012.

[17] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: a system for large-scale machine learning." in *OSDI*, 2016.

[18] J. Zhang, K. Zhou, P. Huang, X. He, Z. Xiao, B. Cheng, Y. Ji, and Y. Wang, "Transfer learning based failure prediction for minority disks in large data centers of heterogeneous disk systems," in *ICPP*, 2019, pp. 66:1–66:10.