Supervised Distributed Hashing for Large-Scale Multimedia Retrieval

Deming Zhai, Member, IEEE, Xianming Liu, Member, IEEE, Xiangyang Ji, Member, IEEE, Debin Zhao, Member, IEEE, Shin’ichi Satoh, Senior Member, IEEE, and Wen Gao, Fellow, IEEE

Abstract—Recent years have witnessed the growing popularity of hashing for large-scale multimedia retrieval. Extensive hashing methods have been designed for data stored in a single machine, that is, centralized hashing. In many real-world applications, however, the large-scale data are often distributed across different locations, servers, or sites. Although hashing for distributed data can be implemented by assembling all distributed data together as a whole dataset in theory, it usually leads to prohibitive computation, communication, and storage costs in practice. Up to now, only a few methods were tailored for distributed hashing, which are all unsupervised approaches. In this paper, we propose an efficient and effective method called supervised distributed hashing (SupDisH), which learns discriminative hash functions by leveraging the semantic label information in a distributed manner. Specifically, we cast the distributed hashing problem into the framework of classification, where the learned binary codes are expected to be distinct enough for semantic retrieval. By introducing auxiliary variables, the distributed model is then separated into a set of decentralized subproblems with consistency constraints, which can be solved in parallel on each vertex of the distributed network. As such, we can obtain high-quality distinctive unbiased binary codes and consistent hash functions with low computational complexity, which facilitate tackling large-scale multimedia retrieval tasks involving distributed datasets. Experimental evaluations on three large-scale datasets show that SupDisH is competitive to centralized hashing methods and outperforms the state-of-the-art unsupervised distributed method significantly.

Index Terms—Hash function learning, large-scale distributed data, multimedia retrieval, supervised distributed hashing.

Manuscript received December 6, 2016; revised May 31, 2017 and July 19, 2017; accepted August 1, 2017. Date of publication September 4, 2017; date of current version February 14, 2018. This work was supported in part by the Major State Basis Research Development Program of China (973 Program) under Grant 2015CB351804; in part by the Natural Science Foundation of China under Grants 61502122, 61672193, 61671188, and 61571164; and in part by the Fundamental Research Funds for the Central Universities under Grant HIT.NSRIF.201653. The associate editor coordinating the review of this manuscript and approving it for publication was Jingkuan Song. (Corresponding author: Xianming Liu.)

D. Zhai, X. Liu, and D. Zhao are with the School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China (e-mail: zhaideming@hit.edu.cn; cxm@hit.edu.cn; dbzhao@hit.edu.cn).

X. Ji is with the Department of Automation, Tsinghua University, Beijing 100084, China (e-mail: xyji@tsinghua.edu.cn).

S. Satoh is with the National Institute of Informatics, Tokyo 101-8430, Japan (e-mail: satoh@nii.ac.jp).

W. Gao is with the National Engineering Laboratory for Video Technology and the Key Laboratory of Machine Perception (MoE), School of Electrical Engineering and Computer Science, Peking University, Beijing 100871, China (e-mail: wgao@pku.edu.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TMM.2017.2749160

I. INTRODUCTION

WITH the surprisingly rapid development of the internet and social media over the past decades, there has been an explosive growing of multimedia content on the web, such as photo sharing site Flickr and social media site Facebook. To make good use of such data, similarity search, a.k.a. nearest neighbor search becomes a critical module in any multimedia retrieval system. Given a query sample, the similarity search is to find one or more nearest neighbors of the query from a database according to some chosen similarity measure [1].

Although great technical progress has been achieved in multimedia retrieval, the state-of-the-art performance is still far from satisfactory. One crucial issue of multimedia retrieval systems is their scalability. In general, the time complexity of exhaustive nearest neighbor search is $O(n^2)$ since all $n$ samples in the target database are scanned. However, when the data size $n$ is very large, it becomes infeasible to compare the query with each sample in the database due to the prohibitive computation and storage costs. To address the scalability issue of large-scale data, several approximate similarity search techniques have been proposed, which can be generally classified into two categories, namely, tree-based approaches and hashing-based approaches. Tree-based approaches organize data with tree structures, while hashing-based approaches map data into bins such that collisions in hash table reflect nearest neighbor relationships. Although tree-based approaches work well on low-dimensional data, they may degenerate to brute-force linear search as the dimension of data increases. Besides, tree-based approaches cost high storage space since the size of tree structure is usually bigger than the data itself. Hashing-based approaches are more appealing. They index data with binary hash codes, which enjoy not only compactness of representation but also low complexity in distance computation. By utilizing hash codes, fast similarity search can be achieved with constant $O(1)$ or sub-linear $O(\log(n))$ time complexity [2], with greatly decreased storage cost.

In recent years, hashing-based approaches have attracted considerable interest for large-scale similarity search. The success of hashing-based methods depends critically on the quality of the used hash functions. Early exploration of hashing is data-independent [3], [4]. The seminal works—locality-sensitive hashing (LSH) [3] and its extensions [4], [5]—generate embeddings for hash functions via random projections, which are independent of the training data. LSH-related methods achieve asymptotic theoretical properties that data points with high
Fig. 1. (a) Distributed databases across ten different locations or sites; (b) distributed databases in (a) can be modeled as a network via a connected undirected graph, where each vertex denotes one database, and each edge denotes that the two related vertices are neighbors with low communication cost.

similarity will have high probability to be mapped to the same hash codes. However, these methods require long codes to achieve good precision, which will result in low recall as the collision probability decreases exponentially with the code length. Recent endeavors aim at data-dependent hashing by employing machine learning techniques to learn the hash function for specific datasets. This new direction is also referred to as hash function learning (HFL) [6]. Some representative HFL works include semantic hashing [7], spectral hashing [8], self-taught hashing [9], binary reconstruction embedding (BRE) [10], minimal loss hashing [11], semi-supervised hashing (SSH) [12], [13], anchor graph hashing [14], optimized kernel hashing [15], kernel-based supervised hashing [16], deep hashing [17], [18], topology preserving hashing [19], robust hashing [20], neighborhood discriminant hashing [21], parametric local hashing [22], [23] and supervised hashing with pseudo labels [24], etc.

Existing HFL methods have been applied in a wide range of applications with great success [25]–[27]. Nevertheless, almost all existing methods are designed for data stored in a single machine, i.e., centralized hashing. Nowadays, with the emergence of more and more large-scale datasets in real-world applications, as illustrated in Fig. 1(a), the data is often distributed across different locations, servers or sites. For instance, a multimedia search engine may perform image/video queries across datasets distributed in multiple locations that are connected by a data communication network [28]. Besides, many datasets in practice are inherently distributed, such as those in applications of mobile surveillance [29] and sensor networks [30]. As a consequence, distributed hashing, which aims at learning the hash function for distributed data, is a worthwhile direction to explore. This motivates the study reported in this paper.

Developing an effective and efficient distributed hashing algorithm is a challenging problem, especially in the following three aspects:

1) It is non-trivial to directly apply existing centralized hashing methods on distributed data to achieve fast similarity search. Although in theory distributed hash function can be learned by assembling all data together as a whole large-scale data, in practice it usually leads to prohibitive computation, communication, and storage costs.

2) Distributed hash function learning should guarantee that the learned hash functions are consistent across different vertices of the distributed network, and the binary codes of the whole data are unbiased.

3) It is non-trivial to efficiently address the optimization problem in distributed hash function learning.

In the literature, to our knowledge, there are a few works [31]–[33] that were tailored for distributed hash function learning. Bahmani et al. first put forward distributed locality sensitive hashing (DisLSH) [31], which is an extension of existing data-independent unsupervised LSH method. Recently, two data-dependent distributed hashing methods called DisITQ [32] and hashing on distributed data (DisH) [33] were proposed. In essence, both DisITQ and DisH extend a previous unsupervised hashing algorithm called iterative quantization (ITQ) [34] to the distributed setting. Specifically, DisITQ is based on the MapReduce strategy [35] to train hash functions; while DisH is to conduct hash function learning on local data of each vertex and then exchange the learned hash functions with its neighboring vertices. Both of them work without exploiting any supervised information, such as label information. Such an unsupervised distributed learning manner limits their performance.

In this paper, we propose a new method called Supervised Distributed Hashing (SupDisH), which learns the hash functions by leveraging the supervised information in a distributed setting. In fact, supervised information plays an important role in classification and retrieval tasks. It has been proved in [36] that the number of labeled data samples has an exponential effect on reducing the classification and retrieval errors. As a consequence, supervised distributed hashing is appealing for the potential of achieving more promising performance than its unsupervised counterparts [31]–[33]. To take advantage of the supervised information in distributed setting, we first cast the distributed HFL into the framework of classification, where the learned binary codes are expected to be distinctive enough for semantic retrieval. More specifically, a joint learning objective function is formulated by integrating hash function
learning and classifier training. Then the distributed supervised hashing model is separated into a set of decentralized subproblems with consistency constraints. At last, the overall optimization problem is iteratively solved by an alternating procedure including three subproblems. Each subproblem can be addressed in parallel in a distributed manner:

1) For the NP-hard binary code learning subproblem, discrete cyclic coordinate descent (DCCD) is devised to solve the discrete optimization problem with a closed-form solution. The binary hash codes are generated bit by bit.

2) For the hash function and classifier learning subproblems, alternating direction method of multipliers (ADMM) [37], [38] is exploited to obtain high quality distinctive unbiased binary code with low computational cost.

To the best of our knowledge, the proposed method is the first work to address supervised distributed hash function learning problem in the literature.

The rest of this paper is organized as follows. In Sections II and III we describe the proposed supervised distributed hashing method in detail, followed by complexity analysis and discussions in Sections IV and V, respectively. Empirical studies conducted on three large-scale datasets are presented in Section VI. Finally, Section VII gives some concluding remarks.

II. SUPERVISED DISTRIBUTED HASHING FORMULATION

In this section, we present in detail the proposed supervised distributed hashing method, including notations and problem definition, and the objective function formulation.

A. Notations

We first define notations appearing in this paper. Denote dataset by calligraphic uppercase letters like \( \mathcal{X} \). Denote boldface lowercase letters like \( \mathbf{a} \) and boldface uppercase letters like \( \mathbf{A} \) as vectors and matrices, respectively. Moreover, we use \( I \) to denote the identity matrix. \( T_\mathcal{X}(\cdot) \) and \( ||\cdot|| \) to denote the trace of a matrix and the Frobenius norm of a matrix, respectively.

B. Problem Definition

Suppose we have a supervised distributed dataset \( \mathcal{X}_i = \{ \mathcal{X}_i \}_{i=1}^P \) with \( n \) data examples, which are distributed across \( P \) vertices of a network. See Fig. 1(b) for example. On the \( i \)-th vertex of the network, there is a local dataset \( \mathcal{X}_i \) including \( n_i \) data points, each of which locates in a \( d \)-dimensional input feature space including \( c \) classes. We denote \( \mathcal{X}_i \) in matrix form as \( \mathcal{X}_i = (\mathbf{X}_i \in \mathbb{R}^{d \times n_i}, \mathbf{Y}_i \in \mathbb{R}^{c \times n_i}) \), where \( \mathbf{X}_i = \{ \mathbf{x}_j \}_{j=1}^{n_i} \) is data feature matrix and \( \mathbf{Y}_i = \{ \mathbf{y}_{kj} \}_{j=1}^{k} \) is the ground truth label matrix. \( y_{kj} = 1 \) if \( \mathbf{x}_j \) from the \( i \)-th vertex belongs to the class \( k \), and 0 otherwise.

In this paper, we solve the problem of generating \( k \)-bit hash codes of all data points through supervised distributed hash function learning. Specifically, \( \mathbf{B}_i \in \{-1, 1\}^{k \times n_i} \) is defined as the hash code matrix of data points in \( \mathcal{X}_i \). The global consistent hash function \( h(\cdot) \), which maps all data points to their binary hash codes, is defined as:

\[
h(\mathbf{x}_i) = \text{sgn}(\mathbf{C}^T \phi(\mathbf{x}_i)), \forall i \in \{1, \cdots, P\},
\]

(1)

where \( \phi(\cdot) \in \mathbb{R}^{d \times d'} \) denotes the feature mapping function, \( \mathbf{C} \in \mathbb{R}^{d \times k} \) is the hash projection matrix, and \( \text{sgn}(\cdot) \) is the elementwise sign function. In general, we can adopt any suitable feature mapping function as \( \phi(\cdot) \), such as the RBF kernel mapping, to obtain the nonlinear processing ability.

C. Objective Function

There are two principles to design the objective function of supervised distributed HFL. Firstly, for semantic retrieval, a good hash function is expected to be discriminative, i.e., data examples from the same semantic class should be mapped to the same hash bin; otherwise, data examples from different classes should have large hamming distance. Secondly and importantly for distributed setting, the objective function should guarantee that the learned hash functions are consistent across all vertices of the distributed network, and the binary codes of the whole data are unbiased.

Following the first principle, to obtain semantic hash function \( h(\cdot) \), we exploit a multi-class classifier to measure the discriminative capability of the hash codes. More specifically, in a distributed setting (i.e. data stores in \( P \) vertices), the objective function is formulated as a constrained optimization problem:

\[
\min_{\mathbf{B}_i, \mathbf{W}, \mathbf{C}} \sum_{i=1}^{P} ||\mathbf{Y}_i - \mathbf{W}^T \mathbf{B}_i||^2 + \lambda ||\mathbf{W}||^2
\]

s.t. \( \mathbf{B}_i = h(\mathbf{X}_i) = \text{sgn}(\mathbf{C}^T \phi(\mathbf{x}_i)), \forall i \in \{1, \cdots, P\} \).

(2)

where \( \mathbf{W} \in \mathbb{R}^{k \times c} \) is the classification matrix which projects the learned hash codes to the assigned class labels. Note that this objective function is specially designed for hash function learning on \( P \) distributed vertices, which makes it different from the centralized hashing method SDH [39].

The first term in (2) is the \( \ell_2 \) loss function for multi-class classifier; the second term is the maximum margin regularizer, which helps to improve the generalization of the solutions. The parameter \( \lambda \) is used to balance the two terms. In the constraint term, the hash function \( h(\cdot) \) encodes the distributed data points \( \mathbf{X}_i \) by \( k \) bits. This objective function formulates a joint learning framework which integrates binary hash codes learning and classifier training. Meanwhile, the hash function is optimized to fit the generated binary hash codes. The derived optimal hash codes are expected to be discriminative enough for the subsequent image retrieval task.

Note that \( \mathbf{B}_i \) is a local variable, which can be updated in parallel in each vertex; while \( \mathbf{W} \) and \( \mathbf{C} \) are global variables which are shared by all vertices, and hence cannot be learned in a completely distributed manner. Alternatively, motivated by the block splitting algorithm [40], we introduce a set of local auxiliary variables \( \{\mathbf{W}_j\}_{j=1}^{P} \) and \( \{\mathbf{C}_j\}_{j=1}^{P} \), one for each vertex, to substitute the global variables \( \mathbf{W} \) and \( \mathbf{C} \). In this way, \( \mathbf{W}_i \) and \( \mathbf{C}_i \) become local variables like \( \mathbf{B}_i \), and thus we can also learn them in parallel. Considering the second design principle,
we further impose the consistency constraints $W_i = W_j$ and $C_i = C_j$ for $i, j \in \{1, \ldots, P\}$. Finally, we obtain a distributed separable objective function as:

$$\min_{B_i, W_i, C_i} \sum_{i=1}^{P} ||Y_i - W_i^T B_i||^2 + \lambda ||W_i||^2$$

s.t. $B_i = \text{sgn} \left(C_i^T \phi(X_i)\right)$,

$$W_i = W_j, C_i = C_j, j \in \mathcal{N}(i), \forall i \in \{1, \ldots, P\},$$

where $\mathcal{N}(i)$ represents the neighbors of the $i$-th vertex in the distributed network. Benefiting from the transitivity property of a connected graph, we only consider the constraints between local neighbors rather than all the vertices. Thereby, it is amenable to decompose the optimization problem into several equivalent subproblems, which can be solved in parallel in a distributed manner.

Generally speaking, optimization of (3) is still a NP-hard problem, since the hash code solutions are constrained to be binary values. To solve this distributed discrete optimization in a computationally tractable manner, we further relax the hard constraint by Lagrangian regularization [41]. Then the distributed separable optimization problem of (3) becomes:

$$\min_{B_i, W_i, C_i} \sum_{i=1}^{P} ||Y_i - W_i^T B_i||^2 + \lambda ||W_i||^2$$

$$+ \beta \sum_{i=1}^{P} ||B_i - C_i^T \phi(X_i)||^2$$

s.t. $B_i \in \{-1, 1\}^{k \times n_i}$,

$$W_i = W_j, C_i = C_j, j \in \mathcal{N}(i), \forall i \in \{1, \ldots, P\}, $$

where the last term is called discrete fitting error loss, which measures deviation between the binary hash codes and the continuous real-value embeddings; $\beta$ is the penalty parameter. If $\beta$ is imposed a very large value, the solutions of (4) will be close to those of (3) in theory. Therefore, in practice, $\beta$ is set an appropriate value to limit the discrepancy of the additional hard constraint, such that optimization of (4) is more flexible. Therein, we still enforce explicit binary constraint in order to achieve hash codes with good quality. It is worth mentioning that when there is only one vertex in the network, the proposed method will degenerate to the supervised hashing algorithm [39] in a centralized setting.

### III. DISTRIBUTED LEARNING AND OPTIMIZATION

In this section, we consider how to efficiently address the non-convex optimization problem with convex linear constraints formulated in (4). Since it is difficult to optimize all parameters together, we employ an alternating procedure [42] to iteratively solve the optimization problem. Specifically, we sequentially optimize w.r.t. one set of variables while keeping all other variables fixed. The procedure is repeated until convergence or arriving at the maximum iteration number $T$. In what follows, we will introduce in detail the optimization of three subproblems, which can be conducted in parallel on each vertex.

#### A. Step 1: Optimizing w.r.t. $\{B_i\}$ When $\{W_i\}, \{C_i\}$ Are Fixed

When all variables but $\{B_i\}$ are fixed, the objective function in (4) becomes:

$$\min_{B_i} ||Y_i - W_i^T B_i||^2 + \beta ||B_i - C_i^T \phi(X_i)||^2$$

s.t. $B_i \in \{-1, 1\}^{k \times n_i}, \forall i \in \{1, \ldots, P\}.$

After a simple algebra derivation and neglecting the constant terms, optimization w.r.t. hash codes $\{B_i\}$ in (5) becomes:

$$\min_{B_i} Tr \left( B_i^T W_i W_i^T B_i - 2B_i^T R_i \right)$$

s.t. $B_i \in \{-1, 1\}^{k \times n_i}, \forall i \in \{1, \ldots, P\},$ (6)

where $R_i = W_i Y_i + \beta C_i^T \phi(X_i)$. Nevertheless, minimization of (6) is still a non-trivial problem since the objective function is NP-hard due to the binary constraints. It can be observed that if we only learn one-bit hash code at a time, then the optimization problem can be solved with a closed-form solution. Accordingly, we learn $B_i$ bit by bit via the discrete cyclic coordinate descent (DCCD) [39], [43] method.

More specifically, define $b^i_m \in \mathbb{R}^{n_i}$ as the $m$-bit hash code vector for data points $X_i$, which is the $m$ row’s transpose of the hash code matrix $B_i$; define $B_{-m}$ as the submatrix of $B_i$ by deleting the $i$th row. Similarly, we can decompose the matrix $B_i$, $R_i$ and $W_i$ in the following form:

$$B_i = \begin{bmatrix} (b^i_m)^T \\ B^i_{-m} \end{bmatrix}, R_i = \begin{bmatrix} (r^i_m)^T \\ R^i_{-m} \end{bmatrix}, W_i = \begin{bmatrix} (w^i_m)^T \\ W^i_{-m} \end{bmatrix}.$$

The subproblem in (6) can then be rewritten as:

$$Tr \left( B_i^T W_i W_i^T B_i - 2B_i^T R_i \right)$$

$$= Tr \left( b^i_m (B_{-m}^T)^T \right) \left( w^i_m (W_{-m}^T)^T \right)$$

$$\mathbb{E} \left( b^i_m (B_{-m}^T)^T \right) - 2 \mathbb{E} \left( b^i_m (B_{-m}^T)^T \right) \mathbb{E} \left( r^i_m (R_{-m}^T)^T \right)$$

$$= 2 \mathbb{E} \left( w^i_m (W_{-m}^T) B^i_{-m} b^i_m - 2 (r^i_m)^T b^i_m + \text{const.} \right).$$

Here we utilize the fact that the inner product of one bit hash code vector is a constant. By neglecting the constant, we get the following objective function for one-bit hash code learning:

$$\min_{b^i_m} \left( \mathbb{E} \left( w^i_m (W_{-m}^T) B^i_{-m} - (r^i_m)^T \right) b^i_m \right)$$

s.t. $b^i_m \in \{-1, 1\}^k, \forall i \in \{1, \ldots, P\}.$ (8)

As a consequence, the optimal one-bit hash code vector $b^i_m$ can be solved with a closed-form solution:

$$b^i_m = -\text{sgn} \left( \mathbb{E} \left( w^i_m (W_{-m}^T) B^i_{-m} - (r^i_m)^T \right) \right),$$

$$\forall i \in \{1, \ldots, P\}. \quad (9)$$

Note that the optimization works in a successive manner, i.e., each bit is optimized based on the pre-learned $k-1$ bits $B_{-m}$.
In addition, the optimal solution for one-bit hash code vector is derived without any relaxation on the discrete constraint, thanks to the fact that different signs are favored for the first-order minimization problem in (8).

To learn the hash codes for multiple bits, we iteratively update each bit using (9) until the procedure converges with a better quality hash codes. As validated by experiment results, the DCCD optimization stated above usually converges within 5 iterations.

B. Step 2: Optimizing w.r.t. \{W_i\} When \{C_i\}, \{B_i\} Are Fixed

When all variables but \{W_i\} are fixed, the objective function in (4) becomes:

\[
\min_{W_i} \|Y_i - W_i^T B_i\|^2 + \lambda \|W_i\|^2
\]

s.t. \(W_i = W_{j}, j \in N(i), \forall i \in \{1, \ldots, P\}\). \hspace{1cm} (10)

To solve this constrained optimization problem, we exploit the well-known alternating direction method of multipliers (ADMM) [37]. ADMM is a variant of the augmented Lagrangian multipliers, which supports the decomposition of variables and provides superior convergence properties. The augmented Lagrangian multipliers of (10) is formulated as:

\[
\mathcal{J} = \min_{W_i} \sum_{i=1}^{P} \|Y_i - W_i^T B_i\|^2 + \lambda \|W_i\|^2 + \\
\frac{1}{2} \sum_{i=1}^{P} \sum_{j \in N(i)} Tr\left(\Gamma_{i,j}^T (W_i - W_j)\right)
\]

\hspace{1cm} (11)

where \(\Gamma_{i,j}\) is the Lagrangian multiplier for the constraint \(W_i = W_j\); \(\rho\) is the penalty parameter of augmented Lagrangian. ADMM solves the optimization of (11) by repeating the following two steps:

\[
\begin{align*}
&W_i^{(t)} := \arg \min_{W_i} \mathcal{J}^{(t-1)};
&\Gamma_{i,j}^{(t)} := \Gamma_{i,j}^{(t-1)} + \rho \left( W_i^{(t)} - W_j^{(t)} \right),
\end{align*}
\]  

\hspace{1cm} (12)

where the superscript \(t\) and \(t-1\) denote the current and last iterative number respectively.

Despite the elegance of the above update rule, the introduced Lagrangian multipliers are so many that the computational complexity increases largely. In fact, it is easy to find that if \(j \in N(i)\) then \(i \in N(j)\), due to the symmetric property of the undirected graph. Hence, in the ADMM formulated in (11), every constraint is considered twice, i.e., \(W_i = W_j = W_j = W_i\). This observation motivates us to further simplify the Lagrangian multipliers.

Specifically, the third term of (11) can be written as

\[
\sum_{i=1}^{P} \sum_{j \in N(i)} Tr\left(\Gamma_{i,j}^T (W_i - W_j)\right) = \\
\sum_{i=1}^{P} \sum_{j \in N(i)} Tr\left(\Gamma_{i,j}^T W_i\right) - \sum_{i=1}^{P} \sum_{j \in N(i)} Tr\left(\Gamma_{j,i}^T W_i\right),
\]

\hspace{1cm} (13)

in which the order of \(i\) and \(j\) is exchanged in the second term, due to the symmetric property of the undirected graph. In this way, the number of Lagrangian multipliers is reduced to \(P\). The above formulation can be further written as

\[
\sum_{i=1}^{P} \sum_{j \in N(i)} Tr\left(\Gamma_{i,j}^T (W_i - W_j)\right) = \\
\sum_{i=1}^{P} Tr\left(\sum_{j \in N(i)} \left(\Gamma_{i,j} - \Gamma_{j,i}\right)^T W_i\right) = \\
\sum_{i=1}^{P} Tr(\Gamma_i W_i).
\]

\hspace{1cm} (14)

Here we define \(\Gamma_i = \sum_{j \in N(i)} \left(\Gamma_{i,j} - \Gamma_{j,i}\right)^T\) as the new Lagrangian multiplier. The main advantage of the new Lagrangian multipliers is that it can greatly reduce the computational complexity of ADMM.

To derive the update rule of the current iteration for new Lagrangian multipliers, we subtract \(\Gamma_{i,j}\) and \(\Gamma_{j,i}\) between any neighboring nodes according to the definition of \(\Gamma_i\):

\[
\begin{align*}
\Gamma_{i,j}^{(t)} - \Gamma_{j,i}^{(t)} = \left(\Gamma_{i,j}^{(t-1)} + \rho \left( W_i^{(t)} - W_j^{(t)} \right) \right) & - \left(\Gamma_{j,i}^{(t-1)} + \rho \left( W_j^{(t)} - W_i^{(t)} \right) \right) \\
& = \left(\Gamma_{i,j}^{(t-1)} - \Gamma_{j,i}^{(t-1)} \right) + 2\rho \left( W_i^{(t)} - W_j^{(t)} \right).
\end{align*}
\]

\hspace{1cm} (15)

For the update of local variable \(W_i\), we can derive a closed-form solution by setting the derivative of \(\mathcal{J}\) in (11) w.r.t. \(W_i\) to be zero. Finally, the new update rules of ADMM are shown as follows:

\[
\begin{align*}
W_i^{(t)} := & \left(2B_i^T B_i + (\rho + 2\lambda)I\right)^{-1} \\
& \left(2B_i Y_i^T - \Gamma_{i,j}^{(t-1)} + \rho \sum_{j \in N(i)} W_{j}^{(t-1)} \right) \\
\Gamma_{i,j}^{(t)} := & \Gamma_{i,j}^{(t-1)} + 2\rho \sum_{j \in N(i)} \left( W_i^{(t)} - W_j^{(t)} \right).
\end{align*}
\]

\hspace{1cm} (16)

C. Step 3: Optimizing w.r.t. \{C_i\} When \{W_i\}, \{B_i\} Are Fixed

When all variables but \{C_i\} are fixed, the objective function in (4) becomes:

\[
\min_{C_i} \|B_i - C_i^T \phi(X_i)\|^2
\]

s.t. \(C_i = C_{j,i}, j \in N(i), \forall i \in \{1, \ldots, P\}\). \hspace{1cm} (17)

To efficiently solve this optimization problem, we still employ ADMM with simplified Lagrangian multipliers. Based on a similar derivation as Step 2, ADMM solves the optimization
Algorithm 1: Supervised Distributed Hashing

Input:
    Supervised distributed data: $X = \{X_i\}_{i=1}^P$, $X_i = \{X_{ij}, Y_{ij}\}$,
    Regularization parameters: $\lambda, \beta, \rho$,
    Maximum iteration number: $T$,
    Code length: $k$.

Output:
    Hash projection $\{C_i\}_{i=1}^P$ and binary codes $\{B_i\}_{i=1}^P$.

Procedure:
1. Initialize $\{B_i\}_{i=1}^P$, $\{C_i\}_{i=1}^P$ and $\{W_i\}_{i=1}^P$, and set $\{A_i\}_{i=1}^P = \{\Gamma_i\}_{i=1}^P = I$;
2. Loop until convergence or reach $T$ times
   2.1. Optimizing w.r.t. $\{B_i\}$ when $\{W_i\}, \{C_i\}$ are fixed.
       Learn $\{B_i\}_{i=1}^P$ bit by bit according to (9); until convergence
   2.2. repeat
       Share $\{W_i\}_{i=1}^P$ with neighboring vertices;
       Update $\{W_i\}_{i=1}^P, \{\Gamma_i\}_{i=1}^P$ with (16); until convergence
   2.3. repeat
       Share $\{C_i\}_{i=1}^P$ with neighboring vertices;
       Update $\{C_i\}_{i=1}^P, \{A_i\}_{i=1}^P$ with (18); until convergence

end loop

of (17) by repeating the following two steps:

$$
\begin{align*}
    C_i^{(t)} &:= (2\phi(X_i)\phi(X_i)^T + \rho I)^{-1} \\
    \lambda_i^{(t)} &:= \lambda_i^{(t-1)} + 2\rho \sum_{j \in N(i)} (C_j^{(t-1)} - C_j^{(t)}) \quad (18)
\end{align*}
$$

Here, $A$ is the augmented Lagrangian multiplier for the constraint $C_i = C_j$.

D. The Algorithm Flow

The working flow of the proposed supervised distributed hashing algorithm is shown in Algorithm 1. It is worth noting that the update of local $\{B_i\}$ (step 2.1), $\{W_i\}, \{\Gamma_i\}$ (step 2.2) and $\{C_i\}, \{A_i\}$ (step 2.3) can be implemented in parallel on each vertex.Benefiting from this property, the proposed method is able to work in a distributed setting efficiently. Thanks to the ADMM method, only the temporary classification matrices and local hash projections are exchanged with its neighboring vertices (step 2.2 and 2.3), which can be conducted with very low communication cost. Moreover, the update of ADMM in each iteration and the computation of one-bit hash code be solved very efficiently with closed-form solutions. As a consequence, the proposed supervised distributed learning algorithm works efficiently.

IV. Complexity Analysis

In this section, we analyze the communication and computational complexity of the proposed SupDisH method. Recall that $d'$ denotes the data dimension after feature mapping; $n_i$ is the data size in the $i$-th vertex; $c$ and $k$ indicate the number of classes and the length of hash code, respectively.

A. Communication Complexity Analysis

In ADMM, to obtain consistent optimization solutions, each vertex needs to share local variables with its neighboring vertices. Supposing the $i$-th vertex has $l_i$ neighbors, the communication complexity of sharing $C_i$ and $W_i$ are $O(l_i d' k)$ and $O(l_i k c)$, respectively. As a consequence, the overall communication complexity of each vertex is $O(l_i (d' k + k c))$, which is independent of the data size $n_i$.

B. Computational Complexity Analysis

The computational cost of the proposed algorithm includes three main parts: one-bit hash code generation (step 2.1), classifier training (step 2.2) and hash function learning (step 2.3). For one-bit hash code generation, the time complexity is $O(c(k - 1)n_i)$. For classifier training and hash function learning, the computational burden lies in the operation of matrix inverse in (16) and (18). For a matrix $A \in \mathbb{R}^{d' \times d'}$, the time complexity of computing its inverse is $T(d') = O(d'^2 376)$ with the method of Coppersmith and Winograd [44]. Accordingly, the computational cost of step 2.2 and 2.3 are $O((k)^2(n_i + c) + (k)^2 376 + kn_i c)$ and $O((d')^2(n_i + k) + (d')^2 376 + d'n_i k)$, respectively. In a nutshell, the computational time complexity linearly increases with the number of local data on each vertex. Since each subproblem can be implemented in parallel on each vertex, the proposed method can be performed more efficiently than centralized hashing algorithms.

V. DISCUSSIONS

The proposed SupDisH method can be alternatively understood from the perspective of multi-task learning [45]. In our algorithm, local optimization on each vertex can be regarded as a single task. Due to the consistency constraints on local variables, the relationships between single tasks on different vertices have positive correlations, with task similarity as one. Therefore, with the help of related tasks on other vertices, the performance of the overall learning can be improved.

Moreover, our method can be deemed a parallel extension of traditional supervised hashing algorithms, which induces another interesting application of the proposed method—reduce the training time of centralized supervised hashing. Specifically, we can divide the whole database of centralized hashing into several subsets and then optimize them in parallel with consistency constraints. Since the size of data assigned to each vertex is smaller, the corresponding training process will be faster.

Besides, the proposed algorithm is flexible, which does not rely on any assumption on data distribution of each vertex in the distributed network. In particular, the proposed SupDisH can
handle scenarios that in some classes data samples are missing on a vertex of the distributed network. This is due to the fact that missing labels do not affect the definition of the ground truth label matrix $Y_i$ and the following optimization procedure. More specifically, according to the definition of the ground truth label matrix $Y_i$, for local set $X_i$, $y_{ki} = 1$ if $x_i$ from the $i$-th vertex belongs to the class $k$, and $0$ otherwise. Suppose data samples of the class $k$ are missing on the $i$-th vertex, then all elements in the $k$-th row of $Y_i$ are zero. The following optimization works in the same way as the scenario that data with full class labels on each vertex.

It is worth noting that although pairwise similarity has been employed in many hashing methods [8], [14], [51], it is not so straightforward as expected to be extended for distributed data. The difficulty lies in the fact that in distributed hashing relationships between any pair of data are involved in optimization for pairwise similarity. Thus it is non-trivial to derive separable objective functions for optimization, which becomes the bottleneck of the distributed hashing algorithms. Our method offers an efficient strategy to conduct optimizations in parallel by directly utilizing the label information in the framework of classification.

VI. EXPERIMENTAL STUDY

In this section, extensive experimental results are provided to evaluate the proposed supervised distributed hashing (SupDisH) method in both retrieval performance and computational efficiency. We assume the data is stored across different vertices in a distributed network. Specifically, we randomly construct a network with 10 vertices, as shown in Fig. 1(b). The implementation of our supervised distributed system is based on the distributed computing engine of MATLAB in Linux.

A. Datasets

The experiments are conducted on three benchmark large-scale image datasets for image retrieval tasks: CIRAR10 [46], NUS-WIDE [47], and MNIST8M [48].

CIFAR10 is a labeled subset of 80 M tiny images [49]. It is composed of 60 K color images of resolution $32 \times 32$ pixels in 10 classes, including airplane, automobile, bird and so on, i.e., each class contains about 6k samples. All these images are represented by 512-dimensional GIST feature descriptors [50]. The whole dataset is split into a query set with 10 K samples, and a distributed dataset with all remaining samples.

NUS-WIDE [47] is a real-world web image dataset which includes images collected from Flickr. It contains 186 577 images with 10 classes. Each image is represented by a 500-dimensional SIFT codebook. We use 99% of the data to form the distributed dataset and the remaining 1% as the query set.

MNIST8M consists of 8 million gray-scale handwritten digit images with $28 \times 28$ pixels from ‘0’ to ‘9’. By default each image is represented by a 784-dimensional vector consisting of its pixel intensities. All the data samples are normalized to have unit length. We randomly select 1% data as the query set and the remaining samples to form the distributed dataset.

B. Compared Methods and Experimental Settings

We compare the proposed SupDisH method against several state-of-the-art centralized and distributed hashing approaches, including the unsupervised methods: 1) PCA-ITQ [34], 2) AGH [14]; the supervised methods: 3) KSH [16], 4) Fash-Hash [52], 5) SDH_anchor [39]; and a state-of-the-art supervised distributed method: 6) DisH [33]. We also compare a variant of SupDisH called SupDisH_local, which learns the hash functions independently on each vertex without any consistency constraint, as formulated in (4). The source codes of the compared methods are all kindly provided by their authors. We tune parameters of these methods to guarantee that the best performances are shown.

The retrieval performance is evaluated in terms of both hash lookup (precision of Hamming radius 2) and mean Average Precision (mAP). For each query and a set of $R$ retrieved data, we first compute the Average Precision (AP) defined as: $AP = \frac{1}{L} \sum_{r=1}^{R} P(r) \delta(r)$, where $L$ is the number of true neighbors in the retrieved set, $P(r)$ denotes the precision of top $r$ retrieved data, and $\delta(r) = 1$ if the $r$-th retrieved data is a true neighbor and $\delta(r) = 0$ otherwise. We then average the AP values over all the queries to obtain the mAP measure. The larger the mAP, the better the retrieval performance. In our experiments, we set $R = 50$. Moreover, we report the precision-recall curves and precision curves by varying the Hamming radius of the retrieved points.

In our method, we compute the hash codes for a new query with (1). In the ideal case, the resulting local hash functions $C_i$’s by our method should be the same since they are learned with consistency constraints. However, ADMM is not theoretically guaranteed to be global optimal [37]. Thereby, we just assign one of local solutions from $\{C_i\}^P_{i=1}$ as the final hash function $C$ and embed all data with it. When a new query $q$ arrives, it is embedded to binary hash codes as $b = \text{sgn}(C^T q)$. Subsequently, the Hamming distances between $b$ and data on all vertices are calculated.

In our method, the feature mapping function $\phi(\cdot)$ is defined as a $d'$-dimensional column vector obtained by the RBF kernel mapping: $\phi(x) = [\exp(||x - a_1||^2/\sigma), \ldots, \exp(||x - a_{d'}||^2/\sigma)]$, where $\{a_i\}$ are the randomly selected $d'$ anchor points from the distributed dataset and $\sigma$ is the kernel width. In our method, we initialize the binary codes for all data with signed random matrix; $C_i$ and $W_i$ with local regression solutions of (17) and (10), respectively. We empirically set $\lambda = 1$, $\beta = 1e-5$, $\rho = 1e-3$; and the maximum iteration $T$ to 10. For AGH, IMH, SDH, and the proposed SupDisH method, we use 1000 randomly sampled anchor points.

C. Comparison With Centralized and Distributed Methods

We compare the proposed SupDisH with both state-of-the-art centralized and distributed hashing methods. For the centralized methods, we assemble all distributed data together into a whole large-scale dataset. Since for MNIST8M the centralized methods lead to prohibitive computation, communication, and storage costs, we instead conduct comparison on its subset MNIST70K, which contains 70 K images. In this case, we
TABLE I
MAP COMPARISON ON CIFAR10

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Method</th>
<th>Code Length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$k = 16$</td>
<td>$k = 32$</td>
</tr>
<tr>
<td>Centralized</td>
<td>PCA-ITQ</td>
<td>0.2776</td>
</tr>
<tr>
<td></td>
<td>AGH</td>
<td>0.3636</td>
</tr>
<tr>
<td></td>
<td>KSH</td>
<td>0.4728</td>
</tr>
<tr>
<td></td>
<td>FashHash</td>
<td>0.4687</td>
</tr>
<tr>
<td>Distributed</td>
<td>SDH anchor</td>
<td>0.5269</td>
</tr>
<tr>
<td></td>
<td>SDH</td>
<td>0.5212</td>
</tr>
<tr>
<td></td>
<td>SupDisH</td>
<td>0.5321</td>
</tr>
</tbody>
</table>

TABLE II
MAP COMPARISON ON MNIST70K

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Method</th>
<th>Code Length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$k = 16$</td>
<td>$k = 32$</td>
</tr>
<tr>
<td>Centralized</td>
<td>PCA-ITQ</td>
<td>0.7576</td>
</tr>
<tr>
<td></td>
<td>AGH</td>
<td>0.6842</td>
</tr>
<tr>
<td></td>
<td>KSH</td>
<td>0.7666</td>
</tr>
<tr>
<td></td>
<td>FashHash</td>
<td>0.7658</td>
</tr>
<tr>
<td></td>
<td>SDH anchor</td>
<td>0.9517</td>
</tr>
<tr>
<td>Distributed</td>
<td>DisH</td>
<td>0.3625</td>
</tr>
<tr>
<td></td>
<td>SupDisH local</td>
<td>0.9343</td>
</tr>
<tr>
<td></td>
<td>SupDisH</td>
<td>0.9503</td>
</tr>
</tbody>
</table>

TABLE III
MAP COMPARISON ON NUS-WIDE

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Method</th>
<th>Code Length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$k = 16$</td>
<td>$k = 32$</td>
</tr>
<tr>
<td>Centralized</td>
<td>PCA-ITQ</td>
<td>0.4350</td>
</tr>
<tr>
<td></td>
<td>AGH</td>
<td>0.4378</td>
</tr>
<tr>
<td></td>
<td>KSH</td>
<td>0.5373</td>
</tr>
<tr>
<td></td>
<td>FashHash</td>
<td>0.5217</td>
</tr>
<tr>
<td></td>
<td>SDH anchor</td>
<td>0.6200</td>
</tr>
<tr>
<td>Distributed</td>
<td>DisH</td>
<td>0.4006</td>
</tr>
<tr>
<td></td>
<td>SupDisH local</td>
<td>0.6407</td>
</tr>
<tr>
<td></td>
<td>SupDisH</td>
<td>0.6417</td>
</tr>
</tbody>
</table>

randomly sample 100 images per class to form a 1 K image query set, and use the rest 69 K images as the distributed dataset.

The mAP and hash lookup results with various code lengths are shown in Tables I–III and Figs. 2–4 for CIFAR10, MNIST70K and NUS-WIDE datasets, respectively. We highlight the highest mAP values in tables. The dashed lines in figures represent the results of distributed methods, while the solid lines indicate the results of centralized methods. For CIFAR10 dataset, as shown in Table I and Fig. 2, in cases of the code length $k = 16$ and $k = 64$, the proposed SupDisH method works even better than SDH anchor and FashHash, which are two state-of-the-art supervised centralized hashing methods; and for other cases, our method is close to the best performed centralized method. For MNIST70K dataset, as shown in Table II and Fig. 3, the performance of our method drops a little compared with SupDisH, but is better than other centralized methods. For NUS-WIDE dataset, as shown in Table III and Fig. 4, in most cases the proposed SupDisH works better than all centralized methods.

Moreover, it can be observed that, SupDisH achieves the best performance in a distributed setting for all cases. Benefiting from the supervised learning, SupDisH significantly outperforms the state-of-the-are distributed hashing method DisH method which is unsupervised. These results demonstrate that SupDisH can indeed obtain more discriminative hash functions in distributed setting.

Besides, SupDisH local, which is a variant of SupDisH, works better than DisH but worse than SupDisH. The comparison results with SupDisH local suggest that jointly optimization with consistency constraints is effective for distributed learning. This observation verifies the analysis from the perspective of multi-task learning in Section V. Compared with
independent optimizations on each vertex, with the help of related learning tasks implemented on other connected vertices, the performance of the overall learning is improved.

The complete precision-recall curves and precision curves on three datasets with various hash bits are illustrated in Figs. 5–7. It can be observed that the curves of SupDisH almost overlap with the second best performed centralized method. In addition, our method outperforms DisH with a large margin. It further verifies that the proposed supervised distributed hashing is more discriminative and can better leverage the supervised information to reduce the semantic gap in a distributed setting.

D. Evaluation on a Very Large-Scale Distributed Dataset

For a very large-scale distributed dataset, the centralized hashing methods—which assemble all distributed data together into a whole large-scale dataset—are infeasible, since they will lead to prohibitive communication, storage and training time costs. In this case, the hash function learning can only be conducted in a distributed setting. In this subsection, we provide comparison results of the proposed SupDisH and its variant SupDisH\textsubscript{local} with the state-of-the-art distributed method DisH [33] and data-independent LSH [3] on a very large-scale distributed dataset MNIST8M.
In our experiments, LSH is able to work in a distributed setting, due to the hash functions are not learned but randomly generated from a locality-sensitive function family. Since the computation of mAP is very slow on a very large-scale dataset, here we only report the mean precision by varying hamming radius from zero to the maximum bit number.

The precision-recall curves and precision curves on MNIST8M are depicted in Fig. 8. We can find that both SupDisH and SupDisH_local consistently outperform the data-independent LSH and unsupervised DisH with a large margin. This is because, the learned hash functions of SupDisH and SupDisH_local that are supervised data-dependent, can explore more information about the semantic structure for specific dataset than randomly generated hash functions in LSH. In addition, the performance improvements of SupDisH_local and SupDisH methods over DisH are significant, which further verify the effectiveness of exploiting supervised information for large-scale semantic retrieval. Generally, SupDisH achieves the highest accuracy, and yields better precision as the code length increases.

### E. Evaluation on Computational Efficiency

In order to demonstrate the time efficiency of our method, we take NUS-WIDE as the test dataset for evaluation. The comparison results of average running time on NUS-WIDE dataset between our SupDisH and three supervised centralized methods are shown in Table IV. We would like to emphasize again that, the three centralized methods need to assemble all distributed data together into a fusion center at first, and then train on the whole large scale dataset. It can be observed that our method has less training time compared with KSH and SDH_anchor, since three subproblems can be conducted in parallel on each vertex in a distributed setting. Our method is competitive to FastHash, which is one of the most efficient hash learning algorithms.

We also test various numbers of vertices in the distributed network to see how training time changes when more vertices are used. Since training time is highly related to the topology of the network when there are the same number of vertices, for fair comparison, in our testing we restrict the network to be the same star topology. Fig. 9 presents the training time comparison when various numbers of vertices involved. We can observe that when more vertices are involved in computation, the training time becomes shorter. This observation also matches with our analysis in computational complexity in Section IV. In a nutshell, distributed (or parallel) learning can lead to improved computational efficiency, and thus has the advantage of scalability for massive data in real world applications.

### F. Convergence Rate Analysis

As stated in Section III, we employ an alternating procedure to iteratively solve the constrained non-convex optimization
problem. Specifically, the first subproblem is addressed by DCCD, which has a closed-form solution. The last two subproblems are addressed by ADMM, which empirically works well even if the objectives are non-convex [53]. To analyze the convergence property, we measure the overall objective function value in (4) and see whether it decreases or not. We choose CIFAR10 as the test dataset in this evaluation, and assume the data is distributed across four vertices in a distributed network. The empirical convergence results on each vertex and the accumulated values of all vertices are reported in Fig. 10. It can be observed that, the objective function values decrease as the iteration number increases, and become stable after about 20 iterations. These results demonstrate that the proposed SupDisH enjoys good convergence property.

VII. CONCLUSION

In this paper, we presented a new supervised distributed hashing (SupDisH) method for large-scale multimedia retrieval. Different from the existing unsupervised distributed hashing works, our method learns discriminative hash functions by leveraging the semantic label information in a distributed manner. To this end, we formulate a joint objective function that integrates binary hash code generation and classifier training, and then extend it to an equivalent separable representation by introducing auxiliary variables and consistency constraints. As such, each subproblem can be solved efficiently in parallel on each vertex with Discrete Cyclic Coordinate Descent (DCCD) and Alternating Direction Method of Multipliers (ADMM). Experimental results demonstrate that the proposed SupDisH algorithm works competitively against the state-of-the-art methods in both scale and accuracy.

In future work, we will explore other learning scenarios, such as semi-supervised distributed learning or deep learning for distributed data to further improve the effectiveness and efficiency of SupDisH.

REFERENCES


Fig. 9. Training time comparison when different number of vertices involved.

Fig. 10. Iterative convergence rates on CIFAR10. (a) Vertex 1. (b) Vertex 2. (c) Vertex 3. (d) Vertex 4. (e) Accumulated of all vertices.


Authors’ photographs and biographies not available at the time of publication.