information

Article

A Ranking-Based Hashing Algorithm Based on the Distributed Spark Platform

Anbang Yang, Jiangbo Qian *, Huahui Chen and Yihong Dong

Faculty of Electrical Engineering and Computer Science, Ningbo University, Ningbo 315211, China; yab119074412@163.com (A.Y.); chenhuahui@nbu.edu.cn (H.C.); dongyihong@nbu.edu.cn (Y.D.)
* Correspondence: qianjiangbo@nbu.edu.cn

Received: 23 January 2020; Accepted: 5 March 2020; Published: 9 March 2020

Abstract: With the rapid development of modern society, generated data has increased exponentially. Finding required data from this huge data pool is an urgent problem that needs to be solved. Hashing technology is widely used in similarity searches of large-scale data. Among them, the ranking-based hashing algorithm has been widely studied due to its accuracy and speed regarding the search results. At present, most ranking-based hashing algorithms construct loss functions by comparing the rank consistency of data in Euclidean and Hamming spaces. However, most of them have high time complexity and long training times, meaning they cannot meet requirements. In order to solve these problems, this paper introduces a distributed Spark framework and implements the ranking-based hashing algorithm in a parallel environment on multiple machines. The experimental results show that the Spark-RLSH (Ranking Listwise Supervision Hashing) can greatly reduce the training time and improve the training efficiency compared with other ranking-based hashing algorithms.

Keywords: similarity search; ranking-based hashing; distributed framework; Spark

1. Introduction

With the continuous development of computing technology and digital media technology in recent years, data generation is increasing every day. This data exists in many forms, including text, images, audio, video, and other forms. Obtaining the information that people need from these massive and high-dimensional data quickly and accurately is an important technical problem [1,2].

At present, there are mainly two ways to solve such problems. One is a tree-based spatial partitioning method, which mainly represents red-black trees, kd-tree, and R-trees [3,4]. However, the disadvantages of this method is that it is only applicable to low-dimensional data. When the dimension rises sharply, it will produce problems such as "dimension disaster", and its search efficiency is close to a linear search function. The other method is a hashing-based search method. This method is also divided into two categories, with one being the data-independent method and the other being locality-sensitive hashing (LSH) [5,6]. This second method employs data dependence. It is also a popular machine learning-based approach that encodes the relevant characteristics of the learning data, thereby improving the retrieval speed and reducing the storage cost [6]. However, some hashing algorithms are currently taking too long to meet the search requirements of the current big data environment.

On the other hand, as the data scale is ever-increasing, the storage and processing requirements of data cannot be met in a stand-alone environment. Therefore, distributed processing systems have emerged for big data, mainly including Hadoop, Storm, and Spark. These distributed systems can process data quickly and in parallel by storing data on multiple computer nodes. By combining the advantages of learning with hashing and distributed systems, this paper designs and implements a distributed learning and hashing method based on the Spark computing platform, which can reduce training time greatly and improve training efficiency.

Section 2 of this paper introduces some of the main learning points for hashing and ranking-based hashing methods. Section 3 introduces the ranking-based hashing algorithm and the resilient distributed dataset model in Spark. Section 4 describes the distributed ranking-based hashing algorithm, along with its design and implementation in the Spark platform. Section 5 analyzes the experimental results of the distributed ranking-based hashing algorithm on several data sets and compares it with several other algorithms. Section 6 is the conclusion.

2. Related Work

In recent years, the learning methods for hashing have been researched extensively due to the faster retrieval speed and lower storage cost, and the core idea is to convert high-dimensional data into compact binary codes by using various machine learning algorithms. By setting a reasonable learning goal, the obtained Hamming codes can maintain the similarity of the original data. The convenient calculation of the Hamming distance also improves the retrieval efficiency of large-scale data.

Currently, the hashing learning methods are mainly divided into two types based on the presence or absence of tag information: unsupervised hashing and supervised hashing (including semi-supervised hashing). Representative unsupervised hashing methods include locality-sensitive hashing (LSH) [7], spectral hashing (SH) [8], self-taught hashing (STH) [9], iterative quantization (ITQ) [10], unsupervised deep video hashing (UDVH) [11], and principal component analysis hashing (PCAH) [12]. The current classical methods of supervised hashing are minimal loss hashing (MLH) [13], supervised hashing with kernels (KSH) [14], supervised discrete hashing (SDH) [15], discrete semantic alignment hashing (DSAH) [16], and linear discriminant analysis hashing (LDAH) [17].

Although the above hashing learning methods have achieved good results, they rarely consider the ranking information in the actual search task. This is because, in general, in the process of searching through the search engine, the returned query results are arranged from top to bottom according to the degree of relevance to the query point.

Based on this, some ranking-based hashing methods have appeared in recent years [18–24], which preserve information by retaining triples (such as learning hash functions using column generation (CGH) [25], Hamming distance metric learning (HDML) [26], etc.) or listwise supervision information (listwise supervision hashing (RSH) [27], ranking preserving hashing (RPH) [28], deep semantic ranking-based hashing (DSRH) [29], discrete semantic ranking hashing (DSeRH) [30], etc.) to learn to generate hashing codes. The triplet supervision method is used to establish a triple $\left(x_q, x_i, x_j\right)$, where x_q represents the query point, x_i represents data similar to the query point, and x_j represents the data that is not similar to the query point. The code is created by establishing the loss function to make x_i more similar to x_q, and x_j less similar to x_q. The listwise supervision method is used to rank all the points in the database in the Euclidean and Hamming spaces, according to the similarity between the query points and data points, and so that the ranking of each point is as consistent as possible in the two spaces.

3. Basic Knowledge

Ranking-Based Hashing Algorithm

The ranking hashing algorithm is one of the supervised learning methods used for hashing algorithms, which better satisfies the search task requirements faced by people in real life. This paper adopts a ranking-based hashing algorithm based on listwise supervision. The basic idea is shown in Figure 1. Here, x_1, x_2, x_3, and x_4 represent four data points in the datasets, and q represents the query point. By calculating the distance between q and four data points in the Euclidean space and then ranking the four points according to the distance, the ranking list $R_1 = (r_1, r_3, r_2, r_4)$ is obtained, where r_1, r_3, r_2, r_4 represent the relevance ranking of x_1, x_3, x_2, x_4 and query point q, respectively. At the same time, by encoding all the data points and calculating the distance between q and the four data points in the Hamming space, the ranking list $R_2 = (r_1, r_2, r_3, r_4)$ is also obtained. Finally, we compare R_1 and R_2, then construct the loss function L to keep R_1 and R_2 as consistent as possible.

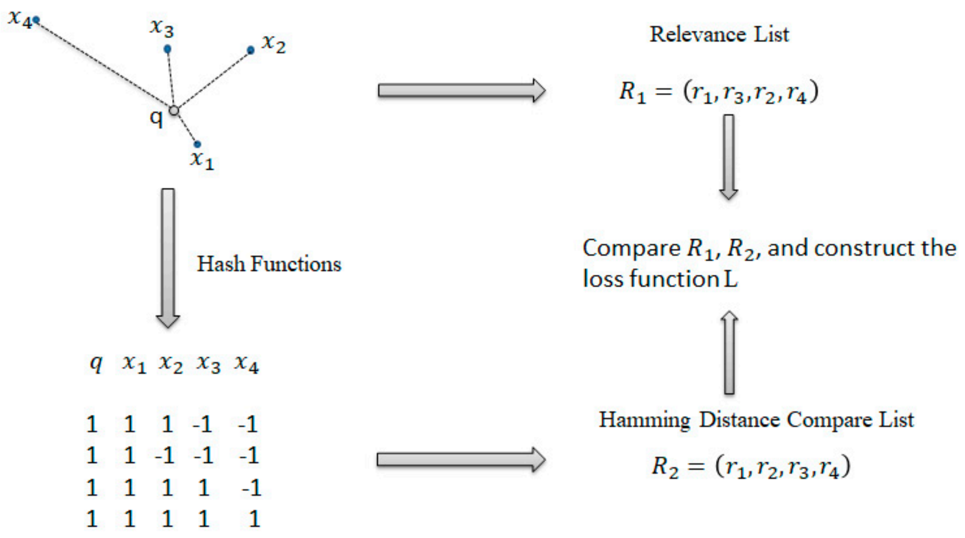

x_4
x_3
x_2
q
x_1
Relevance List
$R_1 = (r_1, r_3, r_2, r_4)$
Hash Functions
Compare R_1, R_2, and construct the loss function L
q x_1 x_2 x_3 x_4
1 1 1 -1 -1
1 1 -1 -1 -1
1 1 1 1 -1
1 1 1 1 1
Hamming Distance Compare List
$R_2 = (r_1, r_2, r_3, r_4)$

Figure 1. Conceptual diagram of listwise supervised ranking-based hashing algorithm.

4. Distributed Ranking-Based Hashing Algorithm

4.1. Overall Description of the Algorithm

At present, the complexity of most of the ranking-based hashing algorithms is too high to meet the existing training requirements, while the distributed Spark platform can execute the algorithm flow in parallel and shorten the training time. Therefore, this paper proposes a ranking-based hashing based on the distributed Spark platform. The algorithm is described as follows:

1. In the distributed Spark environment, all the data in the datasets are mapped to different averaged working nodes. The Euclidean distances between the query points in the query set and all the data points in the datasets in each working node are calculated, and then the distance is ranked. The actual ranking of each point is obtained.
2. Similarly, the query points and all the data points are converted into binary codes on each working node, and the Hamming distance is calculated to obtain the ranking in the Hamming space.
3. According to the loss function, minimize the inconsistency of the data points in the two spaces is minimized. The data transformation matrix on each working node is calculated, and then all the nodes are summed and the average values are calculated using the gradient descent method until the algorithm converges or the number of iterations is reached.

4.2. The Details of the Algorithm

Suppose there are data points in the dataset that expressed as $\chi = \{x_1, x_2, \ldots, x_N\} \in R^{d \times N}$, where d represents the characteristic dimension of the data and there is also a query set $Q = \{q_1, q_2, \ldots, q_M\} \in R^{d \times M}$. The number of nodes in the distributed Spark cluster is S. For any of the working nodes, the dataset χ_s ($\chi_1 \cup \chi_2 \cup \ldots \cup \chi_s = \chi$ and $\chi_i \cap \chi_j = 0$) is assigned, along with the query set Q. For any query point q_j in the query set Q, we can calculate the distance from each of the datasets χ_s directly based on the distance formula of the two points, then ranking the points according to the distance, obtaining the relevance ranking list (we believe that the smaller the Euclidean distance from the query point, the greater the correlation, thus the smaller the ranking), which is recorded as:

$$\mathrm{r}\left(q_j, \chi_s\right) = \left(r_1^j, r_2^j, \ldots, r_N^j\right) \tag{1}$$

where $r_i^j \in \left[1, \frac{N}{S}\right]$, which represents the similarity ranking between the data sample x_i^j and the query point q_j. If $r_m^j < r_n^j$, this means that x_m is more similar to the query point q_j than x_n. Our goal is to obtain the hashing function $f(\cdot)$ to generate the binary codes $h : R \to \{-1, 1\}$ and to define the hashing function mapping as follows:

$$h_i = f(x_i) = \operatorname{sgn}(\mathbf{W}x_i) \tag{2}$$

where $\mathbf{W} \in R^{B\times d}$, B represents the length of the codes, and we define the following formula to calculate the Hamming distance after the query points and the training data points are encoded:

$$\begin{aligned} \operatorname{Ham}\left(h_q, h_i\right) &= \sum_{s=0}^{\frac{B}{\sigma}-1} 2^s \cdot \tfrac{1}{4}\|h_{q_s} - h_{i_s}\|^2 \\ &= \sum_{s=0}^{\frac{B}{\sigma}-1} 2^s \cdot \tfrac{1}{2}\left(\sigma - h_{q_s}^T h_{i_s}\right) \\ &= \sum_{s=0}^{\frac{B}{\sigma}-1} 2^{s-1}\left(\sigma - h_{q_s}^T h_{i_s}\right) \end{aligned} \tag{3}$$

Based on the above formula, we divide the codes into several subspaces of the same length, where the parameter σ represents the length of each subspace. Here, $\frac{B}{\sigma}$ is the number of subspaces that are divided, and then according to the above Hamming distance calculation formula, the ranking of each point in the datasets is obtained. The ranking information of point R_m^j is:

$$\begin{aligned} R_m^j &= 1 + \sum_{k=1}^{N} I\left(\operatorname{Ham}\left(q_j, x_m^j\right) > \operatorname{Ham}\left(q_j, x_k^j\right)\right) \\ &= 1 + \sum_{k=1}^{N} I\left(\operatorname{Ham}\left(q_j, x_m^j\right) - \operatorname{Ham}\left(q_j, x_k^j\right) > 0\right) \\ &= 1 + \sum_{k=1}^{N} I\left(\sum_{s=0}^{\frac{B}{\sigma}-1} 2^{s-1}\left(\sigma - h_{q_{js}}^T h_{x_{ms}^j}\right) - \sum_{s=0}^{\frac{B}{\sigma}-1} 2^{s-1}\left(\sigma - h_{q_{js}}^T h_{x_{ks}^j}\right) > 0\right) \\ &= 1 + \sum_{k=1}^{N} I\left(\sum_{s=0}^{\frac{B}{\sigma}-1} 2^{s-1} \cdot h_{q_{js}}^T\left(h_{x_{ks}^j} - h_{x_{ms}^j}\right) > 0\right) \end{aligned} \tag{4}$$

Finally, we compare the sizes of r_m^j and R_m^j to construct the loss function, so that the two are as consistent as possible.

For any worker node in a spark cluster, we define the loss function as:

$$L(q, x_i, s) = \sum_{i=1}^{N_s} \frac{1}{2\log(1+r_i)}(r_i - R_i)^2 \tag{5}$$

where $\frac{1}{2\log(1+r_i)}$ represents the ranking weight of the data points; obviously, the greater the weight of the real ranking in the Euclidean space, the smaller the value. Therefore, for all nodes, the total loss function is defined as:

$$L(Q, \chi, S) = \sum_{s=1}^{S}\sum_{j=1}^{M}\sum_{i=1}^{N_s} \frac{1}{2\log\left(1 + r_i^j\right)}\left(r_i^j - R_i^j\right)^2 + \frac{\lambda}{2}\|\mathbf{W}\mathbf{W}^\mathbf{T} - \mathbf{I}\|_F^2 \tag{6}$$

where λ represents the balance factor and $\frac{\lambda}{2}\|\mathbf{W}\mathbf{W}^\mathbf{T} - \mathbf{I}\|_F^2$ represents a regularization term to prevent overfitting during training. Finally, we derive the above loss function:

$$\frac{\partial L(Q, \chi, S)}{\partial \mathbf{W}} = \sum_{s=1}^{S}\sum_{j=1}^{M}\sum_{i=1}^{N_s} \frac{1}{\log\left(1 + r_i^j\right)} \cdot \left(R_i^j - r_i^j\right) \cdot \frac{\partial R_i^j}{\partial \mathbf{W}} + 2\lambda\mathbf{W}^\mathbf{T}\left(\mathbf{W}\mathbf{W}^\mathbf{T} - \mathbf{I}\right) \tag{7}$$

The gradient on each working node can be obtained by the updated rule of the gradient descent method:

$$\mathbf{W}_{t+1}^{s} = \mathbf{W}_{t}^{s} - \alpha \frac{\partial \mathrm{L}(q, x_i, s)}{\partial \mathbf{W}_{t}^{s}} \tag{8}$$

where α represents the learning factor, the total of which can be calculated as:

$$\mathbf{W} = \frac{\sum_{s=1}^{S} \mathbf{W_s}}{S} \tag{9}$$

The entire algorithm execution architecture is shown in Figure 2.

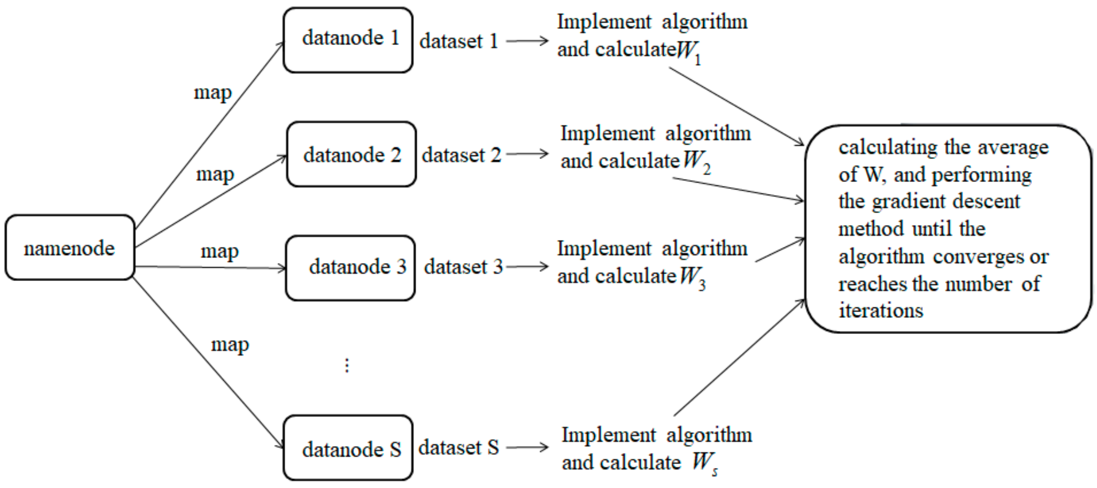

namenode
map
map
map
map
datanode 1
datanode 2
datanode 3
datanode S
dataset 1
dataset 2
dataset 3
dataset S
Implement algorithm and calculate W_1
Implement algorithm and calculate W_2
Implement algorithm and calculate W_3
Implement algorithm and calculate W_s
calculating the average of W, and performing the gradient descent method until the algorithm converges or reaches the number of iterations

Figure 2. The running framework of Spark-RLSH.

5. Experiments

5.1. Experimental Platform Construction

The cluster system of the experimental platform is mainly composed of ten hosts, one of which has a master node (name node) and nine computer nodes (data nodes). The CPU for all hosts is an Intel Core i5-3470, and the memory is 8GB DDR3. In addition, the software environment plays a vital role as a prerequisite for building Hadoop and Spark distributed cluster environments. The specific configuration of the software environment of each machine is shown in Table 1.

Table 1. Configuration of software environment for experimental clusters.

Software Environment	Configuration
Operation System	Ubuntu 14.04
Hadoop	Hadoop 2.7.6
Spark	Spark 2.3.0
Java	jdk-8u172
Python	Python 3.5.2

5.2. Experimental Datasets

This paper mainly experiments with two public datasets, which are the CIFAR-10 dataset and MNIST dataset, and compares these with other ranking-based hashing algorithms (RSH [27], RPH [28]). In the experiment, the data is subjected to pre-processing, such as zero-meanization, and then the image dataset is converted into matrix form by feature extraction and transformation to facilitate the operation.

CIFAR-10 dataset: This contains 60,000 1024-dimensional image data points and includes a total of 10 categories. In this experiment, 320-dimensional gist features were extracted from each photo in the CIFAR-10 dataset, 2000 images were randomly selected as a test dataset, another 2000 images were used as a training dataset, and 200 images were used as a query dataset.

MNIST dataset: This is a handwritten digital image dataset (0–9) containing 60,000 use cases. We also extracted 520-dimensional GIST features for each photo, and randomly selected 2000 images as test datasets. Here, 2000 images are used as a training dataset and 200 images are used as a query dataset.

5.3. Experimental Result

Experiments compare the time required for several hashing algorithms to iterate once when encoding lengths are 16, 32, and 64 bits in two data sets. It can be seen from Figures 3 and 4 and Tables 2 and 3 that both Spark-RLSH datasets have the shortest training times and the fastest running speeds. RSH and RPH have the longest training times due to their operation in the stand-alone environment. At the same time, with the linear increase of the coding length, the training time for Spark-RLSH also grows linearly, while for RSH and RPH, the two ranking-based hashing algorithms, the training time has nothing to do with the length of the code basically remains unchanged.

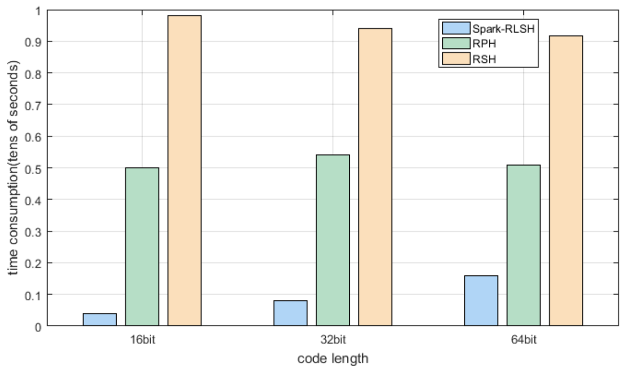


Figure 3. The training time diagram of different length codes for CIFAR-10 datasets.

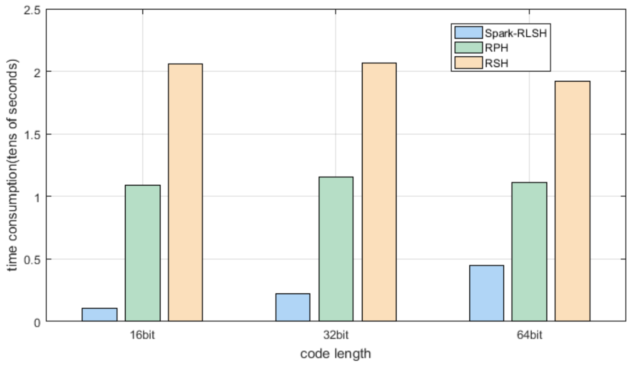


Figure 4. The training time diagram of different length codes for MNIST datasets.

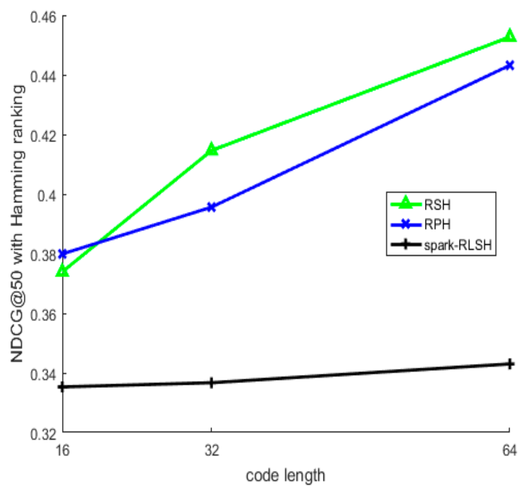
Table 2. Comparison of training times of different algorithms for CIFAR-10 datasets.

	16 bit	**32 bit**	**64 bit**
RPH	5000.68	5420.92	5087.96
RSH	9803.62	9388.89	9162.93
RLSH	26478.66	51835.27	104892.85
Spark-RLSH	395.77	796.71	1602.98

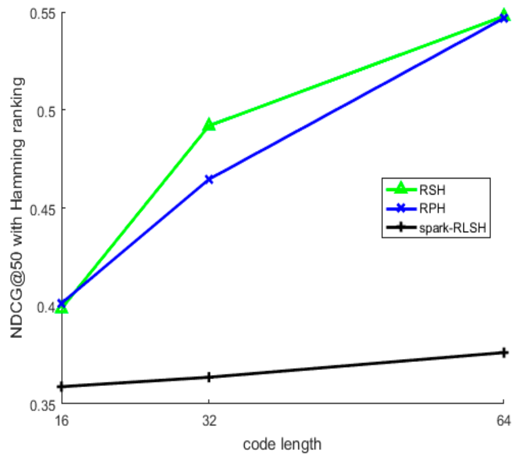
Table 3. Comparison of training times of different algorithms for MNIST datasets.

	16 bit	32 bit	64 bit
RPH	10889.39	11542.78	11095.41
RSH	20577.43	20629.58	19171.19
RLSH	46302.18	93483.59	191638.21
Spark-RLSH	1073.69	2204.01	4495.96

At the same time, Figure 5a,b also compares the top 50 normalized discounted cumulative gain (NDCG) values returned by the three ranking-based hashing algorithms for the two datasets. Although the NDCG value for Spark-RLSH is slightly lower than for RSH and RPH, considering the training time cost, this result is acceptable when the code length is short.



(**a**) CIFAR-10 dataset



(**b**) MNIST dataset

Figure 5. Top 50 normalized discounted cumulative gain (NDCG) values from different datasets. (**a**) CIFAR-10 dataset, (**b**) MNIST dataset.

It can be seen from this result that the ranking-based hashing algorithm implemented on the Spark distributed platform can greatly shorten the training time of the algorithm and improve the training efficiency.

6. Conclusion

This paper introduces the running architecture and working principle of Spark in detail, and proposes the basic principle and algorithm-specific flow of the ranking-based hashing algorithm implemented on the distributed Spark platform. Here, we divide large datasets into small datasets with the same number of working nodes, and compare the ranking of the query set and the small dataset in the Euclidean space and the Hamming space for each working node. Finally, we construct the loss function and run the gradient descent method until the function converges or reaches the number of iterations that minimizes total loss.

Experiments show that the Spark distributed platform can effectively reduce the training time of the model and greatly improve the training efficiency. In the future, we can consider the following points to improve the existing ranking-based hashing algorithm:

1. Improvements in the ranking formula. After converting the data points into binary codes, all the data needs to be ranked according to Hamming distance, and then the ranking list can be constructed. This requires comparison between every two points, so that the time complexity

of the designed ranking algorithm is too high, which can seriously affect the training efficiency. Later, we can consider redesigning the ranking formula to run the algorithm model with the lowest cost.

2. The gradient descent method is implemented as a whole on the distributed platform. This paper considers the complexity of the algorithm in the ranking process. Each working node runs the algorithm model and implements the gradient descent method independently. Although this method can reduce the training time of the model effectively, there is no overall calculation gradient, and there is a certain training error. In the future, the overall comparison of the Hamming distance between the query set and the dataset can be considered, which can improve the accuracy of the search and reduce the training time simultaneously.

Author Contributions: Formal analysis, A.Y. and J.Q.; Funding acquisition, J.Q. and Y.D.; Methodology, J.Q. and H.C.; Software, A.Y; Writing—original draft, A.Y.; Writing—review and editing, A.Y. and J.Q. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by ZHEJIANG NSF(National Science Foundation), grant number LZ20F020001 and LY20F020009, CHINA NSF grant number 61472194 and 61572266, as well as programs sponsored by K.C. Wong Magna Fund in Ningbo University.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Indyk, P. Nearest Neighbors in High-Dimensional Spaces. *Discrete Math. Its Appl.* **2004**, 20042571.
2. Arya, S.; Mount, D.M.; Netanyahu, N.S.; Silverman, R.; Wu, A.Y. An Optimal Algorithm for Approximate Nearest Neighbor Searching. *J. ACM* **1998**, *45*, 91–923. [CrossRef]
3. Guttman, A. R-trees: A Dynamic Index Structure for Spatial Searching. In *ACM SIGMOD International Conference on Management of Data*; SIGMOD: New York, NY, USA, 1984; pp. 47–57.
4. Bentley, J.L. K-d trees for semidynamic point sets. *Annu. Symp.* **1990**, 187–197.
5. Datar, M.; Immorlica, N.; Indyk, P.; Mirrokni, V. Locality-sensitive hashing scheme based on p-stable distributions. *Annu. Symp.* **2004**, 253–262.
6. Wangb, J.; Zhang, T.; Song, J.; Sebe, N.; Shen, H.T. A Survey on Learning to Hash. *IEEE Trans. Pattern Anal. Mach. Intell.* **2018**, *40*, 769–790. [CrossRef] [PubMed]
7. Kulis, B.; Grauman, K. Kernelized Locality-Sensitive Hashing. *IEEE Trans. Pattern Anal. Mach. Intell.* **2011**, *34*, 1092–1104. [CrossRef] [PubMed]
8. Weiss, Y.; Torralba, A.; Fergus, R. Spectral Hashing. Available online: https://people.csail.mit.edu/torralba/publications/spectralhashing.pdf (accessed on 7 March 2020).
9. Zhang, D.; Wang, J.; Cai, D.; Lu, J. Self-taught hashing for fast similarity search. In Proceedings of the 33rd international ACM SIGIR conference, New York, NY, USA, 19 July 2010; pp. 18–25.
10. Fu, H.; Kong, X.; Lu, J. Large-scale image retrieval based on boosting iterative quantization hashing with query-adaptive reranking. *Neurocomputing* **2013**, *122*, 480–489. [CrossRef]
11. Wu, G.; Han, J.; Guo, Y.; Liu, L.; Ding, G.; Ni, Q.; Shao, L. Unsupervised Deep Video Hashing via Balanced Code for Large-Scale Video Retrieval. *IEEE Trans. Image Process.* **2018**, *28*, 1993–2007. [CrossRef] [PubMed]
12. Lin, R.-S.; Ross, D.A.; Yagnik, J. SPEC Hashing: Similarity Preserving Algorithm for Entropy-Based Coding; In Proceedings of the 2010 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), San Francisco, CA, USA2010; pp. 848–854.
13. Norouzi, M.; Blei, D.M. Minimal Loss Hashing for Compact Binary Codes; In Proceedings of the International Machine Learning Society (IMLS), 28 June–2 July 2011; pp. 353–360.
14. Liu, W.; Wang, J.; Ji, R.; Jiang, Y.-G.; Chang, S.-F. Supervised hashing with kernels. In Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, 16–21 June 2012; pp. 2074–2081.
15. Shen, F.; Shen, C.; Liu, W.; Shen, H.T. Supervised Discrete Hashing. In Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, USA, 7–12 June 2015; pp. 37–45.

16. Yao, T.; Kong, X.; Fu, H.; Tian, Q. Discrete Semantic Alignment Hashing for Cross-Media Retrieval. *IEEE Trans. Cybern.* **2019**, 1–12. [CrossRef] [PubMed]
17. Strecha, C.; Bronstein, A.M.; Bronstein, M.M.; Fua, P. LDAHash: Improved Matching with Smaller Descriptors. *IEEE Trans. Pattern Anal. Mach. Intell.* **2011**, *34*, 66–78. [CrossRef] [PubMed]
18. Lin, G.; Shen, C.; Wu, J. Optimizing Ranking Measures for Compact Binary Code Learning. In Proceedings of the 13th European Conference on Computer Vision, Zurich, Switzerland, 6–12 September 2014; Volume 8691, pp. 613–627.
19. Wang, J.; Wangb, J.; Yu, N.; Li, S. Order preserving hashing for approximate nearest neighbor search. In Proceedings of the 21st ACM international conference on Multimedia—MM'13, Nara, Japan, 21 October 2013; pp. 133–142.
20. Ji, T.; Liu, X.; Deng, C.; Huang, L.; Lang, B. Query-Adaptive Hash Code Ranking for Fast Nearest Neighbor Search. In Proceedings of the ACM International Conference on Interactive Experiences for TV and Online Video—TVX'16; Association for Computing Machinery (ACM), Newcastle, UK, 3 November 2014; pp. 1005–1008.
21. Song, D.; Liu, W.; Ji, R.; Meyer, D.; Smith, J.R. Top Rank Supervised Binary Coding for Visual Search. In Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV), Santiago, Chile, 7–13 December 2015; pp. 1922–1930.
22. Song, D.; Liu, W.; Meyer, D.; Tao, D.; Ji, R.D.A.M. Rank Preserving Hashing for Rapid Image Search. In Proceedings of the 2015 Data Compression Conference. In Proceedings of the Institute of Electrical and Electronics Engineers (IEEE), Snowbird, UT, USA, 7–9 April 2015; pp. 353–362.
23. Jiang, Y.-G.; Wang, J.; Xue, X.; Chang, S.-F. Query-Adaptive Image Search with Hash Codes. *IEEE Trans. Multimedia* **2012**, *15*, 442–453. [CrossRef]
24. Zhang, X.; Zhang, L.; Heung-Yeung, S. QsRank:Query-sensitive Hash Code Ranking for Efficient ε-nighbor Search. In Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Providence, RI, USA, 16–21 June 2012.
25. Li, X.; Lin, G.; Shen, C.; Hengel, A.V.D.; Dick, A. Learning Hash Functions Using Column Generation. *arXiv* **2013**, arXiv:1303.0339.
26. Norouzi, M.; Fleet, D.J.; Salakhutdinov, R. Hamming Distance Metric Learning. In Proceedings of the Twenty-sixth Conference on Neural Information Processing Systems, Lake Tahoe, NA, USA, 3–8 December 2012.
27. Wang, J.; Liu, W.; Sun, A.X.; Jiang, Y.-G. Learning Hash Codes with Listwise Supervision. In Proceedings of the 2013 IEEE International Conference on Computer Vision, Sydney, Australia, 2–3 June 2013; pp. 3032–3039.
28. Wang, Q.; Zhang, Z.; Si, L. Ranking Preserving Hashing for Fast Similiarity Search. In Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, Buenos Aires, Argentina, 27 June 2015.
29. Yao, T.; Long, F.; Mei, T.; Rui, Y. Deep Semantic Preserving and Ranking-based Hashing for Image Retrieval. In Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, Buenos Aires, Argentina, 9 July 2016.
30. Liu, L.; Shao, L.; Shen, F.; Yu, M. Discretely Coding Semantic Rank Orders for Supervised Image Hashing. In Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 21–26 July 2017; pp. 5140–5149.