Efficient MapReduce-Based Method for Massive Entity Matching

Pingfu Chao[1,2], Zhu Gao[2], Yuming Li[1,2], Junhua Fang[1,2], Rong Zhang[1,2(✉)], and Aoying Zhou[1,2]

[1] Institute for Data Science and Engineering, New York, USA
{rzhang,ayzhou}@sei.ecnu.edu.cn
[2] Shanghai Key Laboratory of Trustworthy Computing, Shanghai, China
{51121500001,10132510331,51141500019,52131500020}@ecnu.cn

Abstract. Most of the state-of-the-art MapReduce-based entity matching methods inherit traditional Entity Resolution techniques on centralized system and focus on data blocking strategies in order to solve the load balancing problem occurred in distributed environment. In this paper, we propose a MapReduce-based entity matching framework for processing semi-structured and unstructured data. We use a Locality Sensitive Hash (LSH) function to generate low dimensional signatures for high dimensional entities; we introduce a series of random algorithms to ensure that similar signatures will be matched in reduce phase with high probability. Moreover, our framework contains a solution for reducing redundant similarity computation. Experiments show that our approach has a huge advantage on processing speed whilst keeps a high accuracy.

1 Introduction

Entity matching aims to identify entities referring to the same real-world object. However, the rapid growth of web data and User Generated Content (UGC) brings new challenges for entity matching. For instance, in the scenario of C2C (Customer to Customer) online markets, as the rarity of descriptions, missing of uniform schema or intended errors generated by users, tradition entity matching methods are not able to get good match performance.

Though MapReduce provides a new platform for solving massive entity matching problem, new challenges occur: load balancing problem and network transmission cost. Blocking-based entity matching algorithms have been presented to deal with the imbalance problem. Some of the most influential works include sorted neighborhood-based and load-balanced entity matching in Dedoop[3], and document-similarity computation[1]. But for processing non-structured data, these kinds of work meet high network cost and computation cost.

This paper sketches out a random-based framework for entity matching based on MapReduce for semi-structured and unstructured data. Inspired by previous studies, our method expects to reduce both the computation cost and network transmission cost whilst promises the processing performance. We convert high

J. Li and Y. Sun (Eds.): WAIM 2015, LNCS 9098, pp. 494–497, 2015.
DOI: 10.1007/978-3-319-21042-1_48

dimensional entity features into low dimensional bit vector by Locality Sensitive Hash (LSH) function in map phase[4], which reduces the network transmission cost dramatically. We do t rounds of random permutations to those bit vectors. It helps to make similar items paired with high probabilities. Our random-based design can also ensure load-balanced during matching process. Finally, we design a new solution for removing redundant computation in reduce phase.

2 MapReduce-Based Entity Matching Framework

Our entity matching framework is shown in Fig. 1. We represent each entity by its high dimensional feature vector generated from the structured, unstructured or semi-structured description data, if any. These vectors are the input of our MapReduce job, as shown in Fig. 2. The first round of MapReduce job implements the Entity Matching job, while the second one realizes the Redundancy Control.

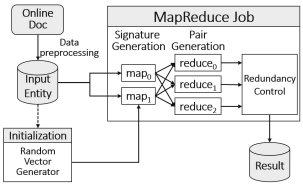

Fig. 1. Framework of random-based entity matching on MapReduce

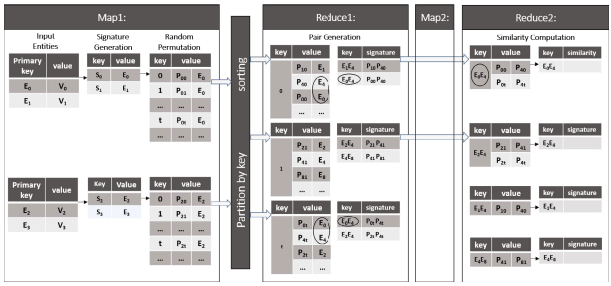

Fig. 2. Example of random-based matching algorithm on MapReduce

Entity Matching. The input is a set of (*key*, *value*) pairs with the entity ID E_u as its key and its k-dimension vector V_u as its value. In map phase, we generate a signature for each item u using the LSH function h_r defined in Eqn.1. We generate a random k-dimension vector set V_r with $|V_r| = d$. Calculating the hash values between u and every vector in V_r using h_r, we get a d-bits vector S_u as the signature for item u, $d \ll k$. Then we apply t rounds of random permutations to every signature S_u and get t different d-dimension bit vectors $\{P_{u1}, P_{u2}, ..., P_{ut}\}$. We regard this result as our map output. So for each entity u, we have t different map outputs as (i, P_{ui}, E_u), in which i refers the permutation series number ($i \in t$), P_{ui} refers the i_{th} permutation result, and E_u is the entity ID.

In Reduce phase, each reducer receives permuted signatures of the same series number. It sorts all signatures and generates pairs between each signature and its m nearest neighbors. Then we calculate the hamming distance of every pair. We output the entity pairs with their similarities as $(E_u E_v, similarity)$ with $u < v$.

$$h_r(u) = \begin{cases} 1 & r.u \geq 0 \\ 0 & r.u < 0 \end{cases} \tag{1}$$

We use the LSH function preserving cosine similarity [2] to generate a signature S_u for each entity u. Since the signature carries most of the characteristics of a vector, we can measure the similarity of two vectors by comparing their signatures. We use the hamming distance between two signatures to represent the similarity, which is reasonable and well proved [4]. In reduce phase, we propose a random permutation algorithm inspired by PLEB algorithm[4] to ensure entities with high similarity to be paired with high probabilities.

Redundancy Control. There can be many duplicated pairs in different groups during reduce phrase as marked in Fig. 2. It may cause significant redundant computation cost. We introduce an extra MapReduce job to reduce duplication. In reduce phase of the first MapReduce job, we remove the similarity computation step, and directly send all the pair-wise data to the second MapReduce job. The second map job does nothing. After the shuffle phase, all pairs with the same entity IDs are grouped together. So we pick one pair of permuted signatures in the group and calculate its hamming distance on behalf of the others. At last, we output the similarity $(E_u E_v, similarity)$ as our result.

3 Experiments

We run experiments on a 22-node HP blade cluster. Each node has two Intel Xeon processors (E5335 2.00GHz) with four cores and one thread per core, 16GB of RAM, and two 1TB hard disks. All nodes run CentOS 6.5, Hadoop 1.2.1, and Java 1.7.0. We use CiteSeerX data set, which contains nearly 1.32 Million citations of total size 2.89 GB in XML format. Each citation includes *record ID, author, title, date, page, volume, publisher, etc* and also *abstract*. We compare the performance of our algorithms with Document Similarity Self-Join (DSSJ)[1] and Dedoop[3]. We use *accuracy* and *run-time* metrics to evaluate performance.

In order to measure the accuracy, we manually generate a validation set which contains 200 records. We output the top 10, 20 and 50 similar pairs for each algorithm. Since Dedoop compares all possible pairs and calculates cosine similarity directly, Dedoop is the best as in Fig.1. Ours achieves better accuracy than DSSJ with much less computation cost as in Fig.3. For processing speed, since Dedoop and DSSJ generate enormous size of pairs, they cost much network transmission and bring big burden for in memory processing as in Fig. 3. In our experiment, the transmission data generated by Dedoop or DSSJ is up to several

Table 1. Accuracy Comparison

Name	Top 10	Top 20	Top 50
DSSJ	90%	95%	94%
Ours	90%	100%	94%
Dedoop	100%	100%	100%

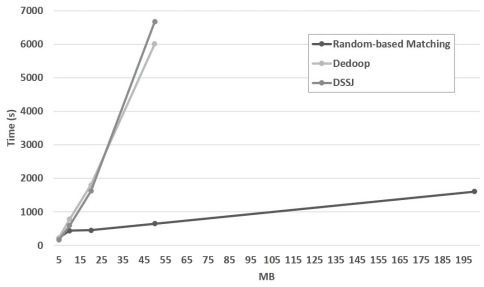

Fig. 3. Run-time Comparison

terabyte for 200MB source data. However, our algorithm is significantly faster than Dedoop, and far more stable even dealing with gigabytes of input data.

4 Conclusion

In this paper, we study the problem of matching the entities with high-dimensional feature vectors based on MapReduce. We take the MapReduce framework as our programming model and point out the two major challenges met on this model, which were load balancing problem and network transmission cost. We propose a random-based matching method to solve the matching problem. We use LSH function to generate signatures for entities and based on random permutations, we can promise similar candidate to be paired with high probabilities. Given the proposed algorithm, we implement it in Hadoop and compare with the other algorithms. We achieve much lower computation cost while still keep high accuracy.

Acknowledgment. This work is partially supported by National Basic Research Program of China (Grant No. 2012CB316200), National Science Foundation of China (Grant No.61232002, 61402180 and 61332006), and Key Program of Natural Science Foundation of Yunnan Province under grant No. 2014FA023.

References

1. Baraglia, R., De Francisci Morales, G., Lucchese, C.: Document similarity self-join with mapreduce. In: Proc. of Data Mining (ICDM), pp. 731–736. IEEE (2010)
2. Charikar, M.S.: Similarity estimation techniques from rounding algorithms. In: Proc. of the Thiry-fourth Annual ACM symposium on Theory of Computing, pp. 380–388. ACM (2002)
3. Kolb, L., Thor, A., Rahm, E.: Dedoop: efficient deduplication with hadoop. Proc. of VLDB **5**(12), 1878–1881 (2012)
4. Ravichandran, D., Pantel, P., Hovy, E.: Randomized algorithms and nlp: using locality sensitive hash function for high speed noun clustering. In: Proc. of ACL, pp. 622–629. Association for Computational Linguistics (2005)