VARIATIONS IN OUTCOME FOR THE SAME MAP REDUCE TRANSITIVE CLOSURE ALGORITHM IMPLEMENTED ON DIFFERENT HADOOP PLATFORMS

Purvi Parmar, MaryEtta Morris, John R. Talburt and Huzaifa F. Syed

Center for Advanced Research in Entity Resolution and Information Quality
University of Arkansas at Little Rock Little Rock, Arkansas, USA

ABSTRACT

This paper describes the outcome of an attempt to implement the same transitive closure (TC) algorithm for Apache MapReduce running on different Apache Hadoop distributions. Apache MapReduce is a software framework used with Apache Hadoop, which has become the de facto standard platform for processing and storing large amounts of data in a distributed computing environment. The research presented here focuses on the variations observed among the results of an efficient iterative transitive closure algorithm when run against different distributed environments. The results from these comparisons were validated against the benchmark results from OYSTER, an open source Entity Resolution system. The experiment results highlighted the inconsistencies that can occur when using the same codebase with different implementations of Map Reduce.

KEYWORDS

Entity Resolution; Hadoop; MapReduce; Transitive Closure; HDFS; Cloudera; Talend

1. INTRODUCTION

1.1. Entity Resolution

Entity Resolution (ER) is the process of determining whether two references to real world objects in an information system refer to the same object or to different objects [1]. Real world objects can be identified by their attributes and by their relationships with other entities [2]. The equivalence of any two references is determined by an ER system based upon the degree to which the values of the attributes of the two references are similar. In order to determine these similarities, ER systems apply a set of Boolean rules to produce a True (link) or False (no link) decision [3]. Once pairs have been discovered, the next step is to generate clusters of all references to the same object.

1.2. Blocking

To reduce the amount of references compared against one another, one or more blocking strategies may be applied [1]. Blocking is the process of dividing records into groups with their most likely matches [4,19]. Match keys are first generated by encoding or transforming a given attribute. Records whose attributes share the same match key value are placed together within a block. Comparisons can then be made among records within the same block. One common method used to generate match keys is Soundex, which encodes strings based on their English pronunciation [5]. Using Soundex, match keys for the names "Stephen" and "Steven" would both have a value of S315. Records containing these strings in the appropriate attribute would have the same match key value, and would thus be placed into the same block for further comparison, along with records with slight differences such as "Stephn" and "Stevn". Blocks are often created by using a combination of multiple match rules. Records can be placed in more than one block, which can sometimes lead to many redundant comparisons [25].

1.3. Transitive Closure

Transitive closure is the process used to discover clusters of references from among matched pairs. The transitive relationship determines that if reference A is equivalent to reference B, reference B is equivalent to reference C, then, by the property of transitivity reference A is equivalent to reference C [2]. This study utilized CC-MR, a transitive closure algorithm developed for use in MapReduce by Seidl, et al. [18] and enhanced in [8].

1.4. Oyster

The ER processes discussed in this paper were performed with OYSTER (Open System for Entity Resolution) Version 3.6.7 [3]. OYSTER is an open source ER system developed by the Center for Advanced Research in Entity Resolution and Information Quality (ERIQ) at the University of Arkansas at Little Rock. OYSTER's source code and documentation is freely available on Bitbucket [7]. The system has proven useful in several research and industry applications [6]. Notably, OYSTER was used in previous studies using large-scale education data in [3] and [5] and electronic medical records in [22].

1.5. Hadoop and MapReduce

Apache Hadoop is an open-source system designed to store and process large amounts of data in a distributed environment [23]. Hadoop is highly scalable, capable of employing thousands of computers to execute tasks in an efficient manner. Each computer, or node, utilizes the Hadoop Distributed File System (HDFS) to store data [9].

Apache MapReduce (MR) is a software framework with the capability to create, schedule, and monitor tasks developed to run on the Hadoop platform [16]. As noted in [10], MR is a good fit for ER operations, as each pairwise comparison is independent of another, and thus can be run in parallel [19,20].

2. BACKGROUND

This research began in an effort to migrate OYSTER's operations to a distributed environment in order to accommodate faster and more efficient processing of larger datasets.

In the Hadoop distributed environment, the preprocessors lack access to a large shared memory space which makes the standard ER blocking approach impossible [11].

This study employs the BlockSplit strategy introduced in [12] to accompany the transitive closure algorithm design introduced in [8]. The algorithm generates clusters by detecting the maximally connected subsets of an undirected graph. BlockSplit is a load balancing mechanism that divides the records in large blocks into multiple reduce tasks for comparison [6]. Three different platforms for distributions of Hadoop were tested to investigate the stability and efficiency of the same TC algorithm.

The expected outcome of the study was that the results from the transitive closure process run in each environment would agree (in terms of F-Measure) with those from the OYSTER baseline run.

3. RESEARCH METHODOLOGY AND EXPERIMENTS

3.1. Transitive Closure Logic

Consistent with the Map-Reduce paradigm, the steps of the algorithm are divided into a Map and Reduce phase. In the Map phase, pairs are generated and identifiers are assigned to each pair. Both the original pair (ex. A,B) and the reverse of each pair (ex. B,A) are generated. Pairs are sorted by the first node and then the second. Pairs that have the same recid as the first element of the pair are placed into a group. Each group is then processed as in the Reduce details below.
Figure 1. Transitive closure logic.

Transitive Closure Logic (Reduce phase)

```
Reduce: Apply Group Processing Rules
(X, Y) represents a generic pair; Set processComplete = True
Get next Group, Examine first pair in the group (X, Y)
If  X <= Y, Then pass the entire group to the output (R1) Else
If groupSize = 1, Ignore (don't pass to output) (R2) Else
Set processComplete = False
For each pair (A, B) following first pair (X, Y) in the group
Create new pair (Y, B)
        Create new pair (B, Y)
            Move (Y, B) to output (R3)
            Move (B, Y) to output (R4)
            Examine last pair of the group (Z,W)
            If X < W
                Move (X, Y) to the output (R5)
After all groups are processed, If processComplete = false
      Make input = output
      Repeat Process Else
      Process complete
Join final output back to original full set of record keys to get singleton
clusters iteratively
New output sorted and grouped AND Iteration Complete with Reduce

Final Output:  Join back to original input and validate with connected
records.
```

3.2. Dataset

To simulate the variety of data quality challenges present in real-world data, a synthetic dataset was used. The dataset contains 151,868 name and address records with 53,467 matching pairs, and 111,181 total clusters. The field names for the dataset use are: recid, fname, lname, address, city, state, zip, ssn, and homephone.

While distributed systems such as Hadoop are designed to handle much larger amounts of data, the dataset size was chosen to accommodate the baseline OYSTER calculations performed on a single local computer.

3.3. Hadoop Distribution Platforms

The details of each Hadoop distribution used for this study are as follows:

- Local HDFS: a stand-alone, single node cluster running Hadoop version 2.8.4.
- Cloudera Enterprise: a multi-node cluster running Cloudera Enterprise version 5.15 (multiple node cluster) along with Apache Hadoop 2.8.4 [13].
- Talend Big Data Sandbox: a single node cluster running Apache Hadoop 2.0 hosted by Amazon Web Services (AWS) [14, 15]

3.4. Boolean Rules

OYSTER (Version 3.6.7) was used to prepare the input data by applying Boolean match rules. The output of this step was a set of indices containing the blocks generated by the Boolean rules. The algorithms used were:

- SCAN: remove all non-alphanumeric characters, convert all letters to uppercase
- Soundex: encode each string based on its English pronunciation.

Table 1 details the rules used in this study. This component of the experiment design was previously used in [5, 21].

Table 1. Boolean Rules Used for each Index.

Index	Rules
1	fname : SCAN lname : SCAN
2	lname: SCAN ssn : SCAN
3	fname: Soundex lname: Soundex ssn: SCAN
4	fname: Soundex lname: SCAN address: SCAN

3.5. Baseline Run

An initial full ER run was conducted using OYSTER to establish a baseline for the dataset. This process included generating a link file containing the pairs considered to be a match, and performing transitive closure to discover the clusters. The expectation was that each MR implementation of TC would produce the same results as the ER calculation in terms of clusters of records. The OYSTER ER Metrics utility [7] was used for evaluation.

3.6. Experiment Steps

The MapReduce transitive closure experiment was conducted in three steps as defined below. The first two steps of the study helped to create benchmarks for the expected results of the TC process, setting the stage for the comparisons made in the final step.

In Step 1, the transitive closure algorithm for MapReduce was run on the Local HDFS cluster, using the pairwise link file that was generated in OYSTER by applying all Boolean match rules. This first run was the benchmark for all further experiments.

In Step 2, separate pairwise link files were generated in OYSTER for each Boolean match rule individually. The files were combined and run through the TC process on the Local HDFS cluster. This step was repeated on the Cloudera platform for validation.

In Step 3, the original source input data is used, and all Boolean rules from Table 1 applied and transitive closure is done on MR [17]. This step was repeated on all three platforms.

3.7. Evaluation

The output from the TC processes on each platform in Step 3 were compared against the initial Step 1 benchmark that was conducted on the Local HDFS cluster with the full match link file. OYSTER's ER Metrics utility [7] was used to compare the results based on the number of true matches found vs. the number of predicted matches. The primary metric used was the F-Measure, which is the harmonic mean of precision and recall [2]. F-Measure is reported with a value from 0 to 1, with 1 meaning a 100% match of the expected results.

4. RESULTS AND DISCUSSION

The Local HDFS cluster executed all steps without issue, and was able to match the benchmark F-Measure value successfully.

The Cloudera Enterprise platform was the most inconsistent among the environments tested. Cloudera was used for Steps 2 and 3. There were multiple attempts required due to compatibility and configuration challenges. The first few runs on the Cloudera platform failed because of a compatibility issue with the custom Soundex algorithm. Subsequent runs were processed with a reduced-size dataset (75, 934 records) which resulted in an F-Measure of 0.68, more than 50% than the full benchmark baseline. After reconfiguration, a final run was done with the full dataset and the subsequent F-Measure was 0.97.

The Talend Big Data Sandbox Hadoop platform was used in Step 3. The initial run's F-Measure was 0.57. After a review of the results, it was determined that the input file format, CSV, caused inconsistencies during processing. This was corrected by converting to a Fixed Width input format as required by the platform. The subsequent run yielded 0.77, over a 10% improvement.

Table 2. Outcomes by platform for Step 3.

Platform	Attempts	Successes	Average Outcome[1]	Best Outcome[1]
Local HDFS	3	2	0.99	0.99
Cloudera	5	2	0.98	0.97
Talend	3	2	0.67	0.77

[1] by F-Measure.

Table 2 summarizes the results of step 3 from all three platforms. "Attempts" is the number of attempts taken, including failed runs. "Successes" are the count of attempts that ran until completion. The Average and Best outcomes of these attempts, according to F Measure, are also listed.

While the same input data and TC algorithm were used, there were a few issues that led to failed attempts and inconsistent results. Working with different platforms required adapting the process to fit the configuration constraints of each system, as with the file format issue mentioned above. In addition, the underlying load balancing mechanisms of each platform seem to vary in the manner in which the pairwise comparison tasks were spread across each node. The platforms used different internal thresholds to determine how the pairs were spread across data processing nodes, which led to inconsistencies among the results, even between iterations on the same platform. This affected the ability of the algorithm to return all matching pairs, which in turn affected the ability to discover the correct amount of clusters during the reduce phase. As previously mentioned, blocking is in itself an ongoing challenge in ER, and distributed environments add to the complexity.

5. CONCLUSION

Although the original intent was to improve the scalability and consistency of the TC algorithm, the experiment results show the inconsistent iterative TC behavior when using different platforms.

The expected result of generating the same matches from using the TC algorithm on MR as the baseline run was impacted by the differences in configuration requirements and blocking behavior on the different Hadoop platforms.

A compromise needs to be made between load balancing and preserving the generated blocks of matched pairs during the Map phase. If matched pairs that should be in the same block are spread across multiple nodes, the Reduce phase cannot correctly and consistently complete the transitive closure process.

These experiments highlight some of the underlying scalability issues for Entity Resolution processes in distributed environments. Future experiments include exploring additional blocking strategies, as well as testing additional platforms and distributed computing frameworks. We also plan to perform this experiment on different datasets to further validate our methods.

ACKNOWLEDGEMENTS

The authors thank the Pilog Group for providing access to their Cloudera environment and Shahnawaz Kapadia for his assistance with the initial research. The authors thank Pradeep Parmar for providing financial support.

REFERENCES

[1] Talburt, J. R., & Zhou, Y. (2013). A practical guide to entity resolution with OYSTER. In Handbook of Data Quality (pp. 235-270). Springer, Berlin, Heidelberg

[2] Zhong, B., & Talburt, J. (2018, December). Using Iterative Computation of Connected Graph Components for Post-Entity Resolution Transitive Closure. In 2018 International Conference on Computational Science and Computational Intelligence (CSCI) (pp. 164-168). IEEE.

[3] Nelson, E. D., & Talburt, J. R. (2011). Entity resolution for longitudinal studies in education using OYSTER. In Proceedings of the International Conference on Information and Knowledge

Engineering (IKE) (p. 1). The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp).

[4] Christen, P. (2012). "A Survey of Indexing Techniques for Scalable Record Linkage and Deduplication," in IEEE Transactions on Knowledge and Data Engineering, vol. 24, no. 9, pp. 1537-1555, Sept. 2012, doi: 10.1109/TKDE.2011.127.

[5] Wang, P., Pullen, D., Talburt, J., & Wu, N. (2015). Applying Phonetic Hash Functions to Improve Record Linking in Student Enrollment Data. In Proceedings of the International Conference on Information and Knowledge Engineering (IKE) (p. 187). The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp).

[6] Osesina, O. I., & Talburt, J. (2012). A Data-Intensive Approach to Named Entity Recognition Combining Contextual and Intrinsic Indicators. International Journal of Business Intelligence Research (IJBIR), 3(1), 55-71. doi:10.4018/jbir.2012010104

[7] OYSTER Open Source Project, https://bitbucket.org/oysterer/oyster/

[8] Kolb, L., Sehili, Z., & Rahm, E. (2014). Iterative computation of connected graph components with MapReduce. Datenbank-Spektrum, 14(2), 107-117.

[9] Manikandan, S. G., & Ravi, S. (2014, October). Big data analysis using Apache Hadoop. In 2014 International Conference on IT Convergence and Security (ICITCS) (pp. 1-4). IEEE.

[10] Kolb, L., Thor, A., & Rahm, E. (2012). Dedoop: Efficient deduplication with hadoop. Proceedings of the VLDB Endowment, 5(12), 1878-1881.

[11] Chen, X, Schallehn, E, Saake, G. (2018). Cloud-Scale Entity Resolution:Current State and Open Challenges, Open Journal of Big Data (OJBD) Volume 4, (Issue 1), (Available at http://www.ronpub.com/ojbd/ ;ISSN 2365-029X).

[12] Kolb, L., Thor, A., & Rahm, E. (2012, April). Load balancing for mapreduce-based entity resolution. In 2012 IEEE 28th international conference on data engineering (pp. 618-629). IEEE.

[13] Cloudera. (2019). Cloudera Enterprise 5.15.x documentation, (available at https://docs.cloudera.com/documentation/enterprise/5-15-x.html); retrieved December 11, 2019

[14] Talend Big Data Sandbox, https://www.talend.com/products/big-data/real-time-big-data/, retrieved December 10, 2019.

[15] Amazon Web Services. (2019)., (available at https://en.wikipedia.org/wiki/Amazon_Web_Services); retrieved November 1,2019.

[16] Salinas, S. O., & Lemus, A. C. (2017). Data warehouse and big data integration. Int. Journal of Comp. Sci. and Inf. Tech, 9(2), 1-17.

[17] Chen, C., Pullen, D., Petty, R. H., & Talburt, J. R. (2015, November). Methodology for Large-Scale Entity Resolution without Pairwise Matching. In 2015 IEEE International Conference on Data Mining Workshop (ICDMW) (pp. 204-210). IEEE.

[18] Thomas Seidl, Brigitte Boden, and Sergej Fries. (2012). CC-MR - finding connected components in huge graphs with MapReduce. In Proceedings of the 2012th European Conference on Machine Learning and Knowledge Discovery in Databases - Volume Part I (ECMLPKDD'12). Springer-Verlag, Berlin, Heidelberg, 458–473.

[19] Hsueh, S. C., Lin, M. Y., & Chiu, Y. C. (2014, January). A load-balanced mapreduce algorithm for blocking-based entity-resolution with multiple keys. In Proceedings of the Twelfth Australasian Symposium on Parallel and Distributed Computing-Volume 152 (pp. 3-9).

[20] Elsayed, T., Lin, J., & Oard, D. W. (2008, June). Pairwise document similarity in large collections with MapReduce. In Proceedings of ACL-08: HLT, Short Papers (pp. 265-268).

[21] Syed, H., Wang, Talburt, J.R., Liu, F., Pullen, D., Wu,N. (2012). Developing and refining matching rules for entity resolution, in Proceedings of the International Conference on Information and knowledge Engineering (IKE), Las Vegas, NV

[22] Gupta T., Deshpande V. (2020) Entity Resolution for Maintaining Electronic Medical Record Using OYSTER. In: Haldorai A., Ramu A., Mohanram S., Onn C. (eds) EAI International Conference on Big Data Innovation for Sustainable Cognitive Computing. EAI/Springer Innovations in Communication and Computing. Springer, Cham.

[23] Muniswamaiah, M., Agerwala, T., and Tappert, C. (2019). Big data in cloud computing review and opportunities. Int. Journal of Comp Sci and Inf. Tech, 11(4), 43-57.

[24] Zhou, Y., & Talburt, J. R. (2014). Strategies for Large-Scale Entity Resolution Based on Inverted Index Data Partitioning. In Yeoh, W., Talburt, J. R., & Zhou, Y. (Ed.), Information Quality and Governance for Business Intelligence (pp. 329-351). IGI Global. http://doi:10.4018/978-1-4666-4892-0.ch017

[25] Efthymiou, Vasilis & Papadakis, George & Papastefanatos, George & Stefanidis, Kostas & Palpanas, Themis. (2017). Parallel Meta-blocking for Scaling Entity Resolution over Big Heterogeneous Data. Information Systems. 65. 137-157.