Summary - Robust Set Reconciliation

Bowen Song[1]
June 5, 2018

***Abstract*— This report is a part of an independent study for a set and string reconciliation problem in a distributed system. The paper[1] studied for this report is considering an alternative to the problem of set reconciliation tolerating an adjustable amount of small errors. The paper refers to the general set reconciliation problem as exact set reconciliation and introduces a solution to situations in a modern distributed system where protocols should ignore small differences.**

I. Introduction

The paper is targeting a general data synchronization situation in the context of set reconciliation problem. The goal is to minimize communication requirement for computing the symmetric difference between two sets with tolerable small differences. The set differences are defined by measurement of Earth's Mover Distance (EMD) as a distance between two point sets. The measurement of two point sets by the EMD is bounded by the minimum coast of matching algorithm for the points in the sets. Some example contributors of set differences include data variety from noise or compression, rounding errors, and privacy-preserving data. These differences, if not considered as seeds of accumulators to large differences, are either considered tolerable or intentionally introduced. By allowing such differences and avoiding *exact* set reconciliation, the protocol arrives at a situational better communication complexity than the information-theoretic lower bound[2], which is just the data representing the exact amount of symmetric differences between two sets. The protocol has a practical computation complexity and is suitable for a one-way communication scheme.

II. Overview

The approach of the algorithm involves random shift, *quadtree* data structure, and Invertible Bloom Lookup Table (IBLT). The random shift plays a role in reducing the probability of failed recovery; The *quadtree* data structure is for grouping the similar data from distance measurement; And IBLT is for compressing set differences to reduce communication complexity.

The algorithm in a two-way communication scheme starts with Host A constructing a message and delivering to Host B for reconciliation. Host A follows the steps of random shift for all set elements by a factor γ and build a *quadtree* of L levels. For each level of the *quadtree*, Host A inserts the elements of the level into an IBLT for compressing data. As the last step, Host A sends the IBLTs and random shift γ factor to Host B for reconciliation.

The reconciliation steps for Host B to extract the symmetric differences between the two sets is very similar to the process of constructing the message. After accepting the message, Host B starts with random shift for the entire set of points in B with the same factor and build a corresponding *quadtree*. Host B decodes the IBLTs from the bottom up to the highest layer with Hamming distance of at most ck, where c is a redundancy factor and k is a bound on the number of symmetric differences to be reconciled. At last, Host B reverse the random shift process to retain the actual elements of symmetric set differences. At this step, the set differences are recovered for both Host A and B. By crosschecking with Host B's set of elements, Host B can distinguish its missing data and the missing data for Host A and send back to Host A to finish the reconciliation process.

III. Invertible Bloom Lookup Table

The IBLT[3] data structure in this algorithm is used to store the entire set of a Host and extract the set symmetric differences on another Host. The general steps for Host A and B to reconcile based on an IBLT is as the following:

1) Host A inserts its entire set into an IBLT
2) Host A sends the IBLT to Host B
3) Host B inserts its entire set into the received IBLT
4) Host B extracts elements of set symmetric differences

In step 1 and 3, a Host inserts all of its elements into two fields based on the XOR of all of its inserted elements through r number of designation hash functions and one fingerprint hash function $f(\cdot)$. The two fields are referred to as *keySum* and *fpSum* respectively. For the designation functions, the number of designation functions generally indicates the number of different locations an element is inserted into the IBLT. In step 3, due to XOR, the insertion removes the same elements from the IBLT, adds the differences from Host B, and, therefore, leaves the symmetric differences in the IBLT after the process.

At last, for the extraction process in step 4, the IBLT exhibits a property that for each field pair of *keySum* and *fpSum*, if the pair satisfies $f(\textit{keySum}) = \textit{fpSum}$, the pair contains one element and is the value of *keySum*. The next step is to reinsert this element to remove this element from the IBLT. In the hope that each element is randomly inserted at r different places, removing an element from the table

[1]B. Song is with Department of Electrical and Computer Engineering, Boston University, Boston MA, sbowen@bu.edu

would leave at least one new field pair that contains only one element to allow further recovery, or else the retrieving process is failed or ended if IBLT is empty. Based on [3], a full retrieving process has a high success rate if the number of field pairs in an IBLT is at least the size of the symmetric differences times a small constant.

IV. Quadtree Data Structure

The algorithm builds a *quadtree* for space where the elements of the set belong to, from the bottom up to avoid constructing empty space. The *quadtree* has its root corresponding to the entire space and its leaves to each individual element in the set. During construction for each level, the Host records the number of elements belonging to each node and stores in a hash table. Since the bottom level contains nodes uniquely corresponding to each element which are the exact information of each set element, the only thing recorded is the counts, which is at most 1 in the bottom level.

V. Performance Metric

Complexity	Communication	Computation
This protocol	$O(\alpha k\, lg(n\Delta^d) lg(\Delta))$	$O(dn\, lg(\Delta))$
Partition-Recon	$O(m\, lg(n))$	$O(mp\, lg(n)(k^2+\alpha))$

α = Redundancy factor
d = Dimension of a set
Δ = number of all d integers in a set
n = Number of elements in a set
k = Fixed number of symmetric differences to be reconciled
lg = Based 2 logarithmic
m = Number of set symmetric differences
p = Partitioning factor

TABLE I
Performance Metric

From Table I, the communication complexity is in the case where an upper bound on the number of symmetric differences to be reconciled is known. The parameter k is a tuning parameter that controls the number of differences to be reconciled. This parameter is usually bounded by the quality of the reconciliation or the communication budget. If k is bounded by the quality of the reconciliation, where quality is measured in terms of EMD distance, k can be iteratively doubled after measuring the quality of a full attempt of reconciliation. This creates a hidden communication and computation complexity of $lg(\text{Number of symmetric differences})$ on top of the stated.

We compare the communication performance of this protocol with that of [4] which optimizes communication complexity for an *exact* set reconciliation problem. We see the performance is linear to the lower bound declared by information-theory[2]. In this protocol, for each combined key, the required number of bits is $O(lg(n\Delta^d))$. To perform better in communication complexity, this protocol has to have its parameter chose to satisfy $\alpha k lg(\Delta)\frac{\Delta^d}{n} < m$. This is possible if the two sets are large and dense with a significant amount of elements under tolerable differences. Fortunately, in a modern distributed system where data evolves mostly by appending instead of editing[5], this protocol would perform well under its limitations.

VI. Conclusion

This protocol is practical in a modern distributed system by targeting applications that tolerate and sometimes expect small set differences. The acquired amount of reduced communication complexity can fit in systems that involve cell phones under its network budget since most of the reconciliation in such system only requires a good enough quality. The protocol has practically experimented over one to multiple dimensional data set reconciliation and image reconciliation. The protocol does seem to have the ability to resume data transfer from the last session after a broken connection since data transferred is a set of isolable IBLTs.

The communication complexity, however, only performs better than existing works for *exact* set reconciliation protocols in the case where the number of tolerable differences is much greater than the number of differences needed to be reconciled. In the case where an upper bound for reconciling the symmetric differences is unknown, the protocol cannot maintain its communication and computation complexity.

References

[1] D. Chen, C. Konrad, K. Yi, W. Yu, and Q. Zhang, "Robust set reconciliation," in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*. ACM, 2014, pp. 135–146.

[2] Y. Minsky, A. Trachtenberg, and R. Zippel, "Set reconciliation with nearly optimal communication complexity," *IEEE Transactions on Information Theory*, vol. 49, no. 9, pp. 2213–2218, Sept 2003.

[3] M. T. Goodrich and M. Mitzenmacher, "Invertible bloom lookup tables," in *Communication, Control, and Computing (Allerton), 2011 49th Annual Allerton Conference on*. IEEE, 2011, pp. 792–799.

[4] Y. Minsky and A. Trachtenberg, "Practical set reconciliation," in *40th Annual Allerton Conference on Communication, Control, and Computing*, vol. 248, 2002.

[5] S. Ghemawat, H. Gobioff, and S.-T. Leung, *The Google file system*. ACM, 2003, vol. 37, no. 5.