Summary - Bandwidth Efficient String Reconciliation using Puzzles

Bowen Song[1]
June 18, 2018

***Abstract*— This report is a part of an independent study for set and string reconciliation problems in distributed systems. The Paper [1] considers string reconciliation problem by reversibly converting it to a set reconciliation problem and performs set reconciliation by an existing protocol [2]. The converting process includes breaking a string into a set of shingles and reverting this process after reconciling the set of substrings. The reverting process includes exhaustively finding Euclidean cycles through a modified version of de Bruijn digraph that is a based on the shingles.**

I. Introduction

The problem of string reconciliation is very similar to that of a set reconciliation; However, the order of the elements within the string matters and is mostly measured by edit distance. A formal definition of string reconciliation problem considers finding and exchanging differences of two strings from two physically separated hosts and achieving a common string under minimum communication and computation complexity. This paper is mainly lowering the communication cost to fit under situations where the network connection is under budget.

The contribution of the paper is a String Reconciliation protocol, referred as *String-Recon*, which converts a string reconciliation problem into a set reconciliation problem through a modified version of de Bruijn digraphs. The *String-Recon* breaks a string into a set of shingles which are character-based n-grams, reconciles the set, and pieces the shingles back to a reconciled string.

The *String-Recon* protocol is compared to three existing works one of which has the communication complexity close to the edit distance times the size of the string. A more elaborate communication cost comparison is presented between *String-Recon* and *rsync* [3], known for its efficiency and a high cost of communication. While the communication complexity does favor the *String-Recon*, there is no mentioning of computation complexity. The *String-Recon* has the best-case performance if all edits are close to each other and for string data with high-entropy, due to the way a string is split. This scenario does apply to many systems with live editing application, where most changes to a string of data are through human editing that causes an edit burst, small edits at a single place.

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

II. Algorithm Overview

The proposed *String-Recon* protocol starts with both hosts Alice and Bob breaking similar strings σ_A and σ_B into shingles, referred as puzzle pieces, which can be considered as elements of sets S_A and S_B. Both Alice and Bob then each computes a modified de Bruijn digraph with the corresponding index of sequential enumeration of a Eulerian cycle in which together represent the original string. Each puzzle piece in each set is then hashed into a unique key by concatenating with the puzzle piece occurrence. At this step, the problem has successfully converted into a set reconciliation problem and is solved by set reconciliation protocol from [2] as an example. At last, Alice and Bob generate modified de Bruijn digraphs from the reconciled two new sets, S'_A and S'_B, and backtrack Eulerian cycles to generate σ'_A and σ'_B. The essential techniques of the protocol such as breaking strings into puzzle pieces, constructing and backtracking a modified de Bruijn digraphs are discussed in the later sections.

III. Breaking String into Set

The *String-Recon* uses *k-shingling* to represent a string as a set of shingles. A string is dot multiplied to a moving mask of all ones with hop size 1 to extract shingles. These shingles can later be converted back to a string though modified de Bruijn digraph which captures shingles overlaps. The length of the mask is directly controlling the length of all shingles. Notice if a mask is very short, there can be multiple ways of reconstructing shingles back to a string, and too long of a mask, while guarantees unique reconstruction, is not breaking the string down enough for an efficient communication cost, since the set reconciliation protocol would be exchanging a very large sized substring for each edit difference.

IV. Modified De Bruijn Digraph

Exhaustively backtracking Eulerian cycles on a modified de Bruijn digraph is suggested as the method to convert a set of shingles back to a string. The modified de Bruijn digraph defined here is a directed graph over an alphabet Σ with vertex substring length size of the moving mask length l_m. Unlike the traditional de Bruijn digraph, the modified version removes unused vertex and ensures at least one Eulerian cycle which results in a string by matching inbound and outbound edges.

V. Performance Metric

The communication complexity for the *String-Recon* is analyzed in detail with a tight upper bound. The communication cost, $O(b + l_m)m + lg(R_A R_B)$ where R is the total number of Euclidean cycles in a modified de Bruijn digraph, is directly related to the length of the moving mask and less affected by the number of Euclidean cycles. While there is no explicit analysis for the computation complexity, it is heavily weighted on the conversion process of converting a set of reconciled substrings back to a reconciled string. The computation is strictly related the process of finding Eulerian cycles from the exhaustive backtracking algorithm and the longer the length of the moving mask the less possible Euclidean cycles exists in a de Bruijn digraph. For this reason, there exists a trade-off between the communication and computation complexity for the *String-Recon* and the tuning parameter is the length of the moving mask.

The comparing experiment between *String-Recon* and *rsync* explores the communication cost with respect to the reconciled string length and edit distance. With a fixed amount of edit distance, *String-Recon* has a communication cost invariant to the increasing total length of the string while *rsync* linearly increases the cost. However, if the total string length is held constant, *rsync* is unaffected by the amount of differences between the strings while *String-Recon* has the communication cost increase linearly to the increasing edit distance. This performance is expected since *String-Recon* has the communication complexity directly linked to string edit distance while *rsync* only considers the full length of the string.

VI. Conclusion

The *String-Recon* protocol requires one round of communication and has a high adaptability for budgeted network connection that relies on existing set reconciliation protocols. However, the computation complexity might not be practical and may fail to reconcile if multiple strings can be decoded from a reconciled shingle set. The protocol while presented in a two-way scheme can be a one-way protocol. The practical use of this protocol would require at least one of the two hosts with enough computation power to perform the exhaustive backtracking algorithm.

The protocol is most suitable to reconcile high-entropy strings with small edit distance and its differences concentrated in one place. The small concentrated editing situation is very similar to the output of live human editing and, therefore, the protocol might be suitable for mobile application to reconcile live document editing.

References

[1] S. Agarwal, V. Chauhan, and A. Trachtenberg, "Bandwidth efficient string reconciliation using puzzles," *IEEE Transactions on Parallel and Distributed Systems*, vol. 17, no. 11, pp. 1217–1225, 2006.

[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] A. Tridgell, "Efficient algorithms for sorting and synchronization," Ph.D. dissertation, Australian National University Canberra, 1999.