Summary - Algorithms for Low-Latency Remote File Synchronization

Bowen Song[1]
August 2, 2018

Abstract— **This report is a part of an independent study for set and string reconciliation problems in distributed systems. The paper [1] considers the problem of file reconciliation in a distributed system using string reconciliation technique. The protocol partitions a file string down to substrings of unfixed-length by *2-way min* [2], a content-dependent string partition technique. The partitions is then fed through a set reconciliation protocol based on CPI approach [3].**

I. Introduction

The remote file synchronization problem considers file update for remote clients from server. To achieve low-latency, a protocol has to restrict number of communication rounds and message size.

II. Algorithm Overview

The paper propose a Low Latency File Synchronization (LLFS) protocol as a single round algorithm with minimum communication cost. The LLFS protocol is designed for file update between a server and a client. The following protocol steps assumes number of symmetrical difference d is known to the client.

1) Both Server and Client use 2-way min method to partition a file into a set of blocks and evaluate a set of hash values from each block
2) Client uses CPI approach [3] with known d to reconcile hash set with that of Server
3) Server merges consecutive blocks not known from Client
4) Server sends the following information for Client to reconstruct the file:
    a) Total number of blocks in the updated file
    b) A bit vector for old and updated file hash sets intersections sorted by value
    c) A bit vector for old and updated file block sets intersections sorted by position
    d) Clients missing blocks
    e) Sequence of the missing blocks
5) Client receives information above and updates local file
    a) Determine intersection file blocks
    b) Arrange missing blocks into local file according to block sequence.

The use of 2-way min file partition technique is explained in Section III. The reason for using hash value is to control the message size for the CPI set reconciliation.

The LLFS chooses CPI approach for its hash set reconciliation as an example to build a single round protocol. Nevertheless, the IBLT would be more efficient in this case where d is known to the client, since the LLFS could benefit from IBLT's computation efficiency.[2]

III. Content-Dependent String Partition

The 2-way min file partition treats a wholesome file as a string and partitions the string into non-fixed sized and non-intersecting blocks. The goal of the partition is to identify the most amount of common blocks between two similar files and redundant data. The steps are as the following:

1) Hash all k-length shingles into an array of integer value kept in shingle position
2) Define each block boundary in front of hash value strictly smaller than preceding and following w values

The w is a turning parameter experimentally proven to be related to average block size. The partition method increases the amount of common partition blocks and block repetition within a file, therefore reduces number of bits required to represent have value of the partition blocks.

IV. Estimating File Partition Block Set Symmetric Difference

The client estimates file partition block set difference d based on random sampling. In each reconciliation process, after file block partitioning, Server sends a set of sample file block hash values to Client to estimate an upper bound on symmetric difference. The size of the sample set is related to file size and chosen by practice, since the partition blocks are content-oriented.

The LLFS protocol can also target a situation where server holds both client's version of file and the updated file to compute an upper bound m on symmetric difference. To know m, Client would need to perform an extra round of communication for m with its file version. However, the assumption of server holding files on all clients requires server to hold all previous versions of the file since different clients may hold different previous versions of the file.

V. Algorithm Performance Analysis

Since the protocol assumes edit distance, m, between the reconciled files is known and is equivalent to block hash set symmetric difference, the theoretical bound on an average-case file reconciliation message size is $O(mlog(n))$, where n is the length of the file.

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

[2]IBLT was not available by 2008 when this paper was published

VI. Conclusion

The LLFS file synchronization protocol harvests file similarity with content dependent block partitioning, reduces communication cost by using a set reconciliation protocol, and estimates file differences by random sampling.

Due to the nature of content-dependent protocol, the protocol parameter is hard to generalize and result is highly content oriented. The file difference estimation based on random sampling is only suitable for similar files with random editing position. Nevertheless, protocol does reduce communication cost for both message size and communication rounds.

References

[1] H. Yan, U. Irmak, and T. Suel, “Algorithms for low-latency remote file synchronization,” in *IEEE INFOCOM 2008 - The 27th Conference on Computer Communications*, April 2008.

[2] D. Teodosiu, N. Bjorner, Y. Gurevich, M. Manasse, and J. Porkka, “Optimizing file replication over limited-bandwidth networks using remote differential compression,” 2006.

[3] 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.