Summary - Practical Set Reconciliation

Bowen Song[1]
May 28, 2018

***Abstract*— This report is a part of an independent study for set and string reconciliation in a distributed system. The report is reflecting a study of set reconciliation extension techniques from [1] extending a protocol based on polynomial interpolation [2] to reduce computation complexity and further customize to modern distributed system situations. The report presents reviews and discussions over these extensions to the set reconciliation protocol from [2].**

I. Introduction

The paper is improving a set reconciliation protocol as described in [2] and will be referred as *Basic-Recon* in the following. The *Basic-Recon* has a linear communication complexity to the number of differences between sets; However, it suffers at least a cubic ordered computation complexity in both situations where upper bound on the number of differences between sets is known or unknown. This paper proposes two extension approaches, *Partition-Recon* and *partition tree*, to remedy the computation complexity problem and adept further to modern distributive system respectively.

II. Overview

The *Basic-Recon* protocol relies on a sufficiently large upper bound for the number of set differences, $\bar{m}$, to perform a successful reconciliation. Since computation and communication complexities are depending on a preset $\bar{m}$, especially computation complexity which is on the cubic order, *Partition-Recon* introduces an approach to keep $\bar{m}$ low and recursively lower the actual number of set differences m to meet the preset $\bar{m}$. This is done by partitioning the set recursively until m is lower than $\bar{m}$. This is an efficient approach compared to setting a large $\bar{m}$ value as suggested in [2].

The protocol further approaches to the application side of a modern distributed system. For example in a Flat Datacenter Storage[3], concurrent read and write are essential parts of its application. The addition of partition tree considers multiple reconciliations on the same set of data.The paper points out a trail of repeated checksum processes performed by the *Partition-Recon* can be removed by saving values of checksums into a *partition tree*.

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

III. Partitioned Set Reconciliation

A partitioned set reconciliation approach is a recursive solution to remedy the large computation complexity of *Basic-Recon*. It uses the notion of divide-and-conquer, recursively partitions the set without intersection and reconciles each partition. The key idea is to keep a small $\bar{m}$ for *Basic-Recon* to process fast and probabilistically try to reduce m in every recursion to meet with the small fixed $\bar{m}$ value for a successful *Basic-Recon* process. The full list of set differences can be obtained by taking the union of recovered sets as the last step. The controlling parameter of *Partition-Recon* includes partitioning factor p, redundancy factor k, and an upper bound for set differences $\bar{m}$. The k and $\bar{m}$ are both parameters inherited from the *Basic-Recon* protocol. The partitioning factor, however, is a new parameter controlling the number of partitions in each recursion. The discussion about how to choose these parameters is included in Section V.

IV. Incremental Partitioned Set Reconciliation

The idea of incremental *Partition-Recon* by the paper is simply recording values of checksums into a partition tree and maintaining by every insertion and deletion to save some computation time. The partition tree maintains internal nodes and leaf nodes correspond to the partitions generated from *Partition-Recon*. An internal node is maintained to represent a partition to have the number of elements intersecting the set and its partition greater than the present bound $\bar{m}$ else is transformed into a leaf node.

V. Performance Metric

Expected Complexity	Communication	Computation
Partition-Recon	$O(mb)$	$O(mpb(\bar{m}^2+k))$
Basic-Recon with $\bar{m}$	$O(b(m+k))$	$O(m^4)$
Basic-Recon without $\bar{m}$	$O(b(m+k))$	$O(m^4)$

k = redundancy factor
b = Length of bitstrings
m = sum of set differences
p = partitioning factor

TABLE I
Performance Metric

From Table I, we can clearly see a lower computation complexity by *Partition-Recon*. Since $\bar{m}$ for *Partition-Recon* is no longer bounded by the actual number of difference

between sets, $\bar{m}$ can be set to a small number and result in an overall linear computation complexity with respect to mb. This is true for the experiments presented by the paper, However, not by a real-world example.

With incremental *Partition-Recon*, we can isolate the checksum process from reconciliation process and perform on insertion and deletion. The computation complexity again is bounded by $\bar{m}$ and logarithmic of the number of elements in a set.

The performance in terms of speed, number of bitstrings reconciled, and redundancy, is within constant factor over optimal communication complexity for incremental *Partition-Recon* with respect to m and p parameters:

Performance	Speed	Redundancy	Round Trips
$\bar{m} \uparrow$	$\downarrow$	$\downarrow$	$\downarrow$
$p \downarrow$	$\downarrow$	$\downarrow$	$\uparrow$

TABLE II
SPEED AND REDUNDANCY TRADE-OFF AND ROUND TRIPS

There exists a trade-off between speed and redundancy as they both going in the same directions from either $\bar{m}$ or p changes. In a distributive system, there can be an optimal spot to maximize overall performance to set the $\bar{m}$ and p values.

VI. CONCLUSION

The two extension approaches greatly reduced the computation complexity for the *Basic-Recon*. The specific complexity reduced is discussed in Section V. The weakness of saving checksums in a partition tree is the obvious extra storage.

Moreover, in a modern distributed system such as the Google File System[4], the most frequent editing is appending new data onto the existing data. This leaves reconciliation process redundant to include the entire set. A more efficient protocol in such system might be benefit from adapting to exchanging the just received information.

The *Partition-Recon* as a recursive divide-and-conquer algorithm reduces the computation complexity of *Basic-Recon* to linear in an expected-case scenario, which was the bottleneck for the *Basic-Recon* protocol. The incremental *Partition-Recon* allows hosts to skip scanning the entire set before reconciling. Both contributions increase the performance of the original *Basic-Recon* protocol without compromising the near optimal communication complexity.

REFERENCES

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

[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] E. B. Nightingale, J. Elson, J. Fan, O. S. Hofmann, J. Howell, and Y. Suzue, "Flat datacenter storage." in *OSDI*, 2012, pp. 1–15.

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