Summary - Efficient Reconciliation and Follow Control for Anti-Entropy Protocols

Bowen Song[1]
August 27, 2018

***Abstract*— This report is a part of an independent study for set and string reconciliation problems in distributed systems. The paper [1] considers reconciling sets of data knowing the content of set symmetrical differences. Based on the anti-entropy gossip protocol, the paper offers novel methods to order value updates and flow control mechanism to distribute network capacity for each reconciling process efficiently.**

I. Introduction

The paper considers an anti-entropy protocol for a distributed system that gossips individual data update until replaced by newer data. The distributed system maintains a set of nodes, each of which can update its own data and hopes to preserve an identical copy of each other. The anti-entropy gossip protocols periodically exchange information between pairs of nodes until all nodes are updated and have the same data. For each data value, the host maintains a locally unique key and version number. Without investigating strategies to identify set differences from two data nodes, the paper focuses on the stage where the exact content of data difference between the nodes are known, and each host has listed these data differences to send to their target host. The proposed algorithm offers data *ordering functions* to send data updates accordingly when the size of the data difference is bigger than the available bandwidth. The paper defines the update rate as the maximum number of updates a node can transmit per gossip interval. A desirable update rate is determined by the number of updates available for a gossip interval. The algorithm assumes that the total update rate for the distributed system is shared among reconciling participants and that flow control mechanism is decentralized, each node self-adjust its update rate and transfer excesses to its peers.

II. Algorithm Overview

The algorithm first assumes that the same amount of bandwidth is assigned to each pair of nodes to reconcile their differences. The goal of the *ordering functions* is to restrain sending additional updates that are obsolete according to a third node in the system. For example, if Alice has values for a key with version 1, Bob has values for the same key with version 2, and Carly has the values with version 3. It would be useless for Bob to update Alice only to be later updated again by Carly.

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

The algorithm saves redundant reconciling updates by prioritizing reconciling values according to *scuttle-breath* and *scuttle-depth* ordering functions. Each data value is preserved in a structure that includes the source of origin, identification key, data value, and version number. The ordering functions take both version number and source of origin into account for data update priority.

A. Assigning Versions to Values

The version number of each value in a node is uniquely assigned in increasing order. Every value renewal caused by non-reconciling activity gets assigned a new version number that is greater than the maximum version number within the node. Upon reconciling, the value only gets updated if its version number is higher than the current version.

B. Scuttle-Breadth ordering function

The *scuttle-breadth* ordering function treats data originated from all nodes fairly by randomly including one value from a node origin at a time. For values originated from the same node, the ordering function prioritizes value with a smaller version number. According to this function, the exact update orders for different nodes should be random to avoid gossip bias.

C. Scuttle-Depth ordering function

The *scuttle-depth* ordering function prioritize updates for values from nodes that have the most amount of updates. These values are usually the least recently updated, and its node of origin has many changes since the last update.

D. Decentralized Flow Control

The flow control addresses the problem of nodes dominating update rate during a pair-wise reconciliation process. In the case where both reconciling nodes desire a higher or lower update rate than the average of the maximum rate, the two participants equally split the update rate between them. However, if only one of the node requires less than the average of maximum update rate, it transfers the excesses to the other node. As a way to self-adjust its update rate, a gossiping node linearly changes its update rate by an increasing and a decreasing constant if data transmission overflows or underflows accordingly.

III. Algorithm Analysis

In order to assign a unique version for each value, the algorithm keeps increasing the maximum version number without reusing any old values; Therefore, the version index can easily overflow. A naive solution to this problem could be subtracting all version numbers by the minimum existing version number in the system periodically.

In addition, reconciling data depends only on version numbers can lead to some unwanted data overwrite. In case two reconciling nodes have an updated value of the same key, the final version of the value would be the one with a higher version number. The value with higher version would come from the node that is more updated at the moment of modification. This intricacy would aggregates if more parties participate in the reconciliation process. The algorithm could improve the current strategy by including timestamps accompanying version numbers and reconciling data based on both information.

IV. Conclusion

The algorithm helps to regulate the performance of anti-entropy gossip protocol by offering two data ordering functions to arrange data differences and reconcile nodes avoiding redundant updates. The decentralized flow control mechanism distributes network capacity equally among the distributed system nodes, transfers excesses to peers with more update requests, and avoids starving nodes with fewer updates.

References

[1] R. Van Renesse, D. Dumitriu, V. Gough, and C. Thomas, "Efficient reconciliation and flow control for anti-entropy protocols," in *proceedings of the 2nd Workshop on Large-Scale Distributed Systems and Middleware*. ACM, 2008, p. 6.