Summary - Synchronizing Namespace with Invertible Bloom Filters

Bowen Song[1]
June 25, 2018

Abstract— **This report is a part of an independent study for set and string reconciliation problems in distributed systems. The paper [1] studies the problem of data synchronization in the named data networking architecture for CCNx applications. The proposed protocol *iSync* synchronize namespaces between hosts before reconciling the actual content. The protocol uses invertible Bloom filter in a two-level tree structure and compares its performance with CNNx synchronization protocol.**

I. Introduction

The Paper considers the problem of set reconciliation for named data networking applications (NDN), where the target of synchronization is a set of namespaces. These namespaces are addresses to where the actual data locates in the attached repositories. While the namespaces are local to its repositories, they each serve as a unique identification to a file. A namespace is, therefore, uniquely referred to a file that is identical in any synchronized repository. The protocol, in an NDN system, is aiming at reconciling differences between sets of namespaces to achieve data synchronization in services such as public key distribution, file sharing, and route distribution.

The iSync is a multi-round protocol that maintains data synchronization with a two-level invertible Bloom filter (IBF) structure. The protocol identifies set differences from general collections at the first level and computes specific item-wise differences at the second level. This protocol is compared to CNNx sync, an existing NDN synchronization protocol, in computation and communication cost. Since the protocol reconciles different hosts using a multi-round communication scheme, a consistent network connection is required.

II. Algorithm Overview

The Paper proposes *iSync* as a synchronization layer for CNNx applications. The *iSync* consists of a repository and a sync agent. The repository holds the actual file content while a sync agent keeps track of file namespaces. On insertion, the protocol partitions the incoming files into collections by their prefixes and each partition can be reconciled independently. The *iSync* targets the efficiency to maintain a digested status of an entire repository, distinguish collection version, and recover set differences.

III. CCNx Synchronization Protocol

The CCNx Synchronization is an interactive set reconciliation protocol that organizes namespaces in *Sync Tree* data structure. By reconciling namespace collections between hosts, CCNx Sync identifies the differences between repositories of the hosts and exchange data package accordingly. A host can have multiple collections and, therefore, multiple *sync trees*. The *sync tree* is a collection-based data structure for namespaces. A tree leaf holds the sum of namespace hashes that belong to the leaf as the key and the list of namespaces as value, while each parent node holds the combined hashes of its children. With this structure, the tree root holds the sum of entire collection hashes.

The general steps for a one-way synchronization process of CCNx Sync start interactively from root of the sync tree to bottom. Alice sends the collection name and the sum of the collection of hash values from the root for Bob to compare. If Bob holds a different value, the synchronization process continues with Bob requesting for further comparison of the sum of hashes of each child of the root for the collection. The process recursively continuous until it reaches one or more leaf nodes with a different sum of hash values. When receiving a request for leaf value comparison, Alice sends out the entire list of the namespace from the leaf node for Bob to compare. Bob singles out the missing items and request Alice to send the file content from the repository.

According to the protocol scheme, the balance of the tree and the size of the leaf node is vital to bring down the cost of looking up and differentiating set differences. The height of the tree determines the number of communication rounds required to reconcile a collection and the number of namespaces within a leaf node control the message size when the protocol reaches to the second final step. While it is not discussed in the paper, we could image the control of the tree shape is adjustable by selecting an appropriate hash function. Nevertheless, due to the nature of this design, there can only be one hash function for each tree to determine the outcome of a sync tree. This brings some limitations and requires low to zero collision rate for hashing each namespace in a collection.

IV. iSync

Similar to the CCNx sync, the iSync is also a collection based interactive protocol that reconciles namespaces. The improvement is within namespace holding data structure where *iSync* constructs a two-level tree of IBLTs [2] to replace the sync tree from the CCNx sync protocol. The root of iSync tree for a collection holds the sum of hashes, also known as a digest, of the entire set and an IBLT holding the sums of hashes of all its children. The second level is the leaf

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

node level, and the number of leaf nodes is unbounded. A leaf holds the sum of namespace hashes as key and an IBLT holding all the item namespaces in this the leaf as value. Since namespaces are hashed into IBLTs, the iSync address item lookup efficiency by employing a hash-indexed table for every leaf node IBLT.

The workflow of the iSync protocol is not very different from CCNx sync. The iSync starts with Alice sending the digest of each collection for Bob to compare. When Bob discovers a difference, he sends a request for the collection. Alice sends the IBLT of the collection containing the digest of all leaf nodes of the collection. Bob subtract its IBLT for the collection with the one from Alice to obtain the differences and request for the IBLTs for those leaves. Alice sends the IBLTs of those leaf nodes for Bob to subtract the differences. Bob, at last, ask for the missing element and Alice sends the actual file from repository over to complete synchronization. The communication rounds are, therefore, fixed by this scheme; however, the protocol is efficient only if the changes to the collections are small enough.

V. Algorithm Performance Analysis

The underlying problem with this protocol is the IBLT construction. Not all collections within a host would have the similar number of items, therefore, the size of IBLT changes during item insertions. Increasing the size of IBLT would cause table subtraction invalid, for example, subtracting a smaller sized IBLT from a double sized IBLT is not going to resolve. Every time the number of items inserted into an IBLT is exceeding what is allowed, the protocol would need to reconstruct a new IBLT, and this is a costly process. On the other hand, initializing an exceedingly large IBLT would increase communication cost per reconciliation and computation cost per insertion.

The iSync addresses the problems mentioned above with periodic sync and backup local IBLTs scheme. The periodic sync controls the number of difference between hosts never gets too large; nevertheless, this solution is still venerable for upload bursts, a user uploading a significant number of small files that creates a namespace insert burst between a sync period. The iSync also keeps a set of private backup IBLTs with a bigger size for each IBLTs which remedies rest of the issues. This backup is potentially a burden for space complexity for each host in the system.

VI. Conclusion

The iSync protocol provides a convenient scheme to address NDN synchronization process by extensively exploring the power of IBLT. According to its experimental results, the computation and communication complexity is strictly lowered in a large-scaled system. Although not mentioned, the number of rounds of communication is strictly reduced as well since iSync has a two-level tree structure.

The protocol still requires multi-rounds of communication and requires a consistent network connection. There are several bottlenecks for the protocol specifically the IBLT size change. Creating a new IBLT due to a dramatic number of items increase can be very expensive, especially in a host with limited resources. The agility of the data structure holding namespace item is necessary for applications that have abrupt changes in the number of namespaces. An alternative to IBLT is *Partition-Recon* [3] which supports the two-level tree structure and is less dependent on the number of changes in a synchronization period to be small. Moreover, it would support reconciling file with a customized order based on hash value.

The iSync protocol is a replacement for CCNx synchronization protocol for a CCNx application under NDN architecture. The iSync exploits IBLT and replace sync tree with a two-level IBLT tree and reduces computation and communication complexity and rounds for a sizeable scaled system. However, the space complexity increases for holding IBLTs for each node, their backup IBLTs, and namespace ID lookup tables.

References

[1] W. Fu, H. B. Abraham, and P. Crowley, “Synchronizing namespaces with invertible bloom filters,” in *Architectures for Networking and Communications Systems (ANCS), 2015 ACM/IEEE Symposium on*. IEEE, 2015, pp. 123–134.

[2] 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.

[3] 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.