Poster – iSync: A High Performance and Scalable Data Synchronization Protocol for Named Data Networking*

Wenliang Fu[1], Hila Ben Abraham[2] and Patrick Crowley[2]
[1]School of Computer Science, Beijing Institute of Technology
[2]Computer Science and Engineering, Washington University in St. Louis
fuwenl@bit.edu.cn, {hila, pcrowley}@wustl.edu

ABSTRACT

This paper presents a high performance synchronization protocol for named data networking (NDN). The protocol, called iSync, uses a two-level invertible Bloom filter (IBF) structure to support efficient data reconciliation. Multiple differences can be found by subtracting a remote IBF from a local IBF, and therefore, from a single round of data exchange. We evaluated iSync's performance by comparing it to the default data synchronization protocol of CCNx. Experiments show that iSync is significantly faster for different network sizes and topologies, and it requires less overhead for synchronizing various file sizes.

Categories and Subject Descriptors

C.2.2 [**computer-communication networks**]: Network Protocols—*Applications*

Keywords

Named data network; high performance; data synchronization; invertible Bloom filters

1. INTRODUCTION

Data synchronization of multiple nodes is a fundamental operation in many Internet applications, such as cloud storage, group communication, and routing protocols. In named data networking (NDN) [1], keeping namespaces synchronized has recently emerged as a basic service required by many NDN applications, from Dropbox-style file sharing to supporting mobile and ad-hoc vehicular communication. NDN also uses a core synchronization protocol to support a key-based trust model which requires public key exchange.

The goal of a synchronization protocol is to keep a dataset (or a *collection*) up-to-date among distributed participants. In other words, a synchronization protocol must replicate a dataset's content among participating hosts. In NDN, a content item is represented by a namespace, so a synchronized NDN collection consists of the content names – a list of namespaces.

Data synchronization consists of three basic tasks: 1) understanding whether a set is up-to-date or out-of-date, 2) finding set differences, and 3) retrieving missing items.

This paper describes iSync, a high performance and scalable data synchronization protocol based on IBFs. iSync uses a two-level IBF structure to support efficient data reconciliation. The first level identifies collections which are out-of-date; the second level discovers the IDs of all the items that exist in a remote collection, but are missing in the local one. This feature allows the difference reconciliation process to efficiently skip collections that have no updates.

2. PROTOCOL DESIGN

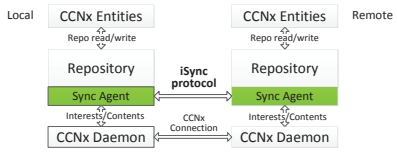

Figure 1: Data Synchronization Model.

As shown in Fig.1, iSync consists of two components: a repository and a sync agent. The repository offers an interface for CCNx entities (i.e., applications) to insert files and publish sync collections. The iSync protocol operates between any two nodes that publish the same collection. In the NDN architecture, a collection is defined by a namespace. All the content items to be synchronized under a published collection must be named using the collection's namespace in their prefix. Upon a content insertion, iSync automatically adds the new content to the local collections whose namespace matches the new item prefix.

The sync agent indexes the inserted files' names and updates a digest that reflects the current contents of each sync collection. It periodically broadcasts local digests, while receiving remote ones. By comparing local with remote digests, the sync agent can identify whether the collections (local and remote) are synchronized. Synchronization starts when a remote collection digest does not match the local collection digest. The set difference of a local and a remote node is found by repeatedly requesting, receiving, and subtracting remote IBF tables from local and global IBF tables. Multiple differences can be found from each subtraction, and therefore, from a single round of data exchange.

*Hila Ben Abraham is the corresponding author of this work.

ICN'14, September 24–26, 2014, Paris, France.
ACM 978-1-4503-3206-4/14/09.
http://dx.doi.org/10.1145/2660129.2660161.

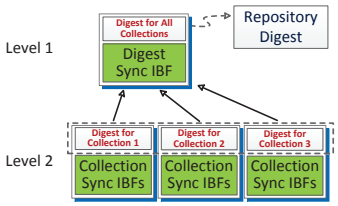

Figure 2: Hierarchical Data Structure.

Fig. 2 shows the hierarchical IBF design. The protocol utilizes a two-level IBF design, *Digest sync IBF* and *collection sync IBFs*, to perform the tasks mentioned above. While level one records the status of the entire repository, level two logs file insertions or deletions of each sync collection separately. An update changes a second level IBF by hashing the content name into the corresponding *collection sync IBF*. This hashing causes a change to the collection digest, and therefore invokes an update in the first level IBF, including the repository digest.

The original design of the IBF handles only fixed-length item names and does not support content lookup very well [2]. To address these limitations, iSync performs two tasks: mapping variable-length file names into fixed-length IDs, and recording what items have been inserted. The former is done by using a hash-indexed table to support bidirectional mapping between file names and IDs. During the entire synchronization process, file names are replaced by fixed-length file IDs. The latter utilizes a counting Bloom filter to support file insertions, deletions, and queries.

The iSync protocol utilizes IBFs to hold the set of name IDs for each sync collection. While it is very efficient to compute differences between two IBFs by subtracting them and decoding the resulting IBF, there is no guarantee that all the differences can be decoded. For a fixed-size IBF, the more updates it holds, the less likely it can be perfectly decoded. To ensure the decoding of all the differences, iSync implements a difference size control mechanism. Hosts that have declared the same sync collections periodically confirm the consistency of their data sets by exchanging their repository digests. This periodic operation guarantees bounded delay of file shares and limits the potential number of differences between the hosts.

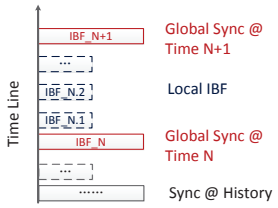

Figure 3: Local and Global IBFs for one Sync Collection.

For each collection, two types of IBFs are used: global and local. A change in a local collection is indexed into the current local IBF. When the number of differences exceeds a defined maximum, iSync creates a new local IBF to store future changes. A global IBF is the local IBF on the time of a sync cycle and is regarded as a public version of the collection data. To find all changes made in a sync cycle, iSync subtracts a remote global IBF from the host's global IBF. If the subtraction fails to obtain the complete set difference, iSync uses the local IBFs between the two latest global IBFs to decode smaller set differences. The combination of those IBF types makes differences between two IBFs traceable. An application can tune the periodic sync time and the IBF size to support its specific requirements.

3. EVALUATION

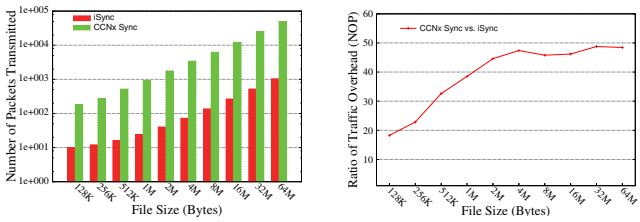

Figure 4: Traffic Overhead for Various File Sizes.

We synchronized files of different sizes (from 128 KB to 64 MB) and measured the traffic overhead by capturing the network traffic. Fig. 4 shows the number of packets transmitted by CCNx Sync [3] and iSync for different file sizes. In terms of the number of packets, iSync is about 18 and 48 times more efficient than CCNx Sync while sharing files of 128 KB and 64 MB respectively.

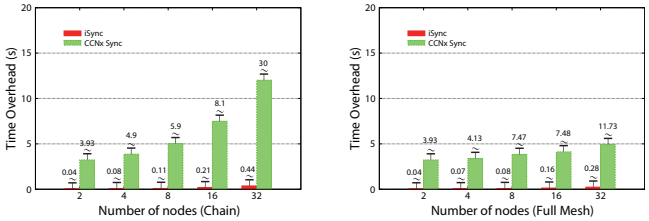

Figure 5: Average Synchronization Time of iSync and CCNx Sync.

We evaluated and compared the performance of the iSync and CCNx Sync protocols in multiple network topologies with a range of nodes scaling from 2 to 32. As shown in Fig. 5, iSync is significantly faster in both chain and full mesh topologies for a range of network sizes.

4. CONCLUSION

This paper presents a high performance synchronization protocol which utilizes the IBF to synchronize published collections between nodes. The protocol uses a two-level IBF design and can reconcile a number of differences in a single comparison. We found that on average, iSync is about 66 times faster than CCNx on a range of network sizes, while it uses 18 to 48 times fewer packets.

5. REFERENCES

[1] Lixia Zhang, Deborah Estrin, and et al. Named data networking project. *Relatório Técnico NDN-0001, Xerox Palo Alto Research Center-PARC*, 2010.

[2] Michael T Goodrich and Michael Mitzenmacher. Invertible bloom lookup tables. In *Communication, Control, and Computing (Allerton), 2011 49th Annual Allerton Conference on*, pages 792–799. IEEE, 2011.

[3] Content centric networking (CCNx) project website. http://www.ccnx.org.