EaSync: A Transparent File Synchronization Service across Multiple Machines

Huajian Mao, Hang Zhang, Xianqiang Bao, Nong Xiao, Weisong Shi, Yutong Lu

EaSync: A Transparent File Synchronization Service across Multiple Machines

Huajian Mao, Hang Zhang, Xianqiang Bao, Nong Xiao, Weisong Shi*, Yutong Lu

State Key Laboratory of High Performance Computing, Changsha, Hunan, China.
National University of Defense Technology, Changsha, Hunan 410073, China.
*Wayne State University, 5057 Woodward Ave, Detroit, MI 48202, USA.
{huajianmao,nongxiao,ytlu}@nudt.edu.cn weisong@wayne.edu

Abstract. In our daily life, people increasingly use multiple machines to do their daily work. As platform switching and file modification are so frequently that a way for file synchronization across multiple machines is required to make the files in synchronized. In this paper, we propose EaSync, a transparent file synchronization service across multiple machines. EaSync proposes several key technologies for file synchronization oriented service, including a timestamp based synchronization protocol, an enhanced deduplication algorithm DS-Dedup. We implement and evaluate the EaSync prototype system. As the result shown, EaSync outperforms other synchronization system in operation latency and other metrics.

Keywords: file synchronization; multiple machines; deduplication; EaSync.

1 Introduction

In our daily life, people increasingly use multiple machines to do their daily work. They intermittently work on different platforms in different places, and nomadically do their routine work like document processing, presenting and so on. During the processing of the files on these machines, files may be created, read, modified and deleted which will make different machines have different views and versions of files and folders. As platform switching and file modification are so frequent that a way for file synchronization across multiple machines will be required to make the files on different work platforms in synchronization.

With diving into how users typically process documents on multiple machines, we find that, the essential requirement for a convenience file synchronization service needs to be satisfied is that it should be pervasively accessible and transparent to the applications, and make nomadic access with intermittent connection always latest guaranteed at any location. However, several challenging issues arise in file synchronization across multiple machines. First, data should be accessed at anywhere; however, the network may be partitioned or client may go offline. Second, service should keep working correctly with multiple clients; however conflicts always show up. Third, data consistency should be assured; however, network is always partitioned,

and it is not easy to keep data consistency in the synchronization service across multiple machines. Forth, synchronization service should be transparent to the applications in the client; however, most of the applications can only do local file operations.

Several new methods like Dropbox[1], UbuntuOne and DBank have been provided as a service for the Internet users. However, Dropbox sometime does not synchronize the files very well, especially when you have multiple person share the files. Also they are all close source project, especially at the server, where much more research issues and large scale data center challenges may be discovered. Also, they only work with the support of connection with their servers, which means, when you only have a local area network connection, the file synchronization will fail, even if the connected machines are actually connected directly.

In this paper, we propose EaSync, a transparent file synchronization service across multiple machines. EaSync enables users to store and sync files online and across multiple computers. EaSync gives users the probability to work anywhere with the freshest data. This paper describes our experience from conception to implementation and evaluation. The main contributions presented in this paper are as follows: First, EaSync states the problem about the file synchronization across multiple machines and make an open source project for both client and server side. Second, several key designs for file synchronization oriented service are proposed in this paper. The designs include a dual-timestamp based synchronization protocol, an enhanced deduplication algorithm DS-Dedup for deduplication. Third, we implement and evaluate the EaSync prototype system. The experiment results show that EaSync outperforms the other platforms in many aspects.

The rest of this paper is organized as follows. We then outline the system model and design of EaSync in detail in Section 2, followed by the implementation in Section 3. We show our experimental results in Section 4. Section 5 presents the prior related work. We conclude our paper and present the future work in Section 6.

2 Design of EaSync

EaSync includes four main components which are EaSync Client, EaSync Gateway, MetaStore, and DataStore. Figure 1 is an overview of EaSync architecture. EaSync client consists of three components which are Local Update Observer (LObsvr), Remote Update Observer (RObsvr), and the synchronization daemon (Syncer). EaSync client runs synchronization protocol with the EaSync Gateway. The synchronization protocol will be discussed in Section 2.2. EaSync client monitors the file update on the client, and starts the synchronization to EaSync Server.

Gateway is the entry of EaSync service. It deals with the requests from both the EaSync clients and the web browser. In order to scale up, load balancing is considered. The main work of the gateway is to route the requests (with Logic Controller) to help the clients to fetch metadata from MetaStore (with a metastore manager instance) and data from DataStore (with a datastore manager instance). MetaStore manages all of the metadata information for the users. Also the synchronization log records are

also managed in MetaStore. Another important component in server side is the DataStore where all of the data are store.

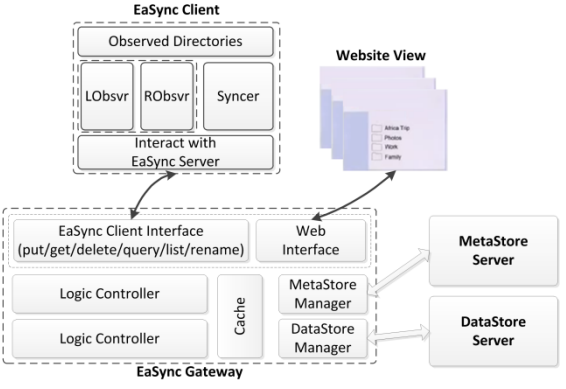

Fig. 1. An overview of EaSync architecture.

2.1 Synchronization Protocol

We propose a hybrid timestamp based synchronization protocol to keep the managed files in sync. In this protocol, two timestamps are used, one is the client-side timestamp as local time, and the other one is server-side one as a global timestamp. The local time is used to record when the update is made, while the global one is used to record when the update is submitted to the EaSync server (by reading the gateway at server side system time).

Figure 2 gives the steps of file synchronization between EaSync client and server. We describe this protocol in detail as following. L.1) The Lobsvr observes the modification of the candidate objects with the notification mechanism. And once an object is modified, client runs step L.2), which inserts a change log into a local to-be-synced pool, and marks the change type to be "changed by local". In this step, the local timestamp is recorded. After this, L.3) is executed by the Syncer. It reads the updates with type of "changed by local", and starts step L.4) to fetch the freshest data in local, and synchronizes it to the server with step L.5).

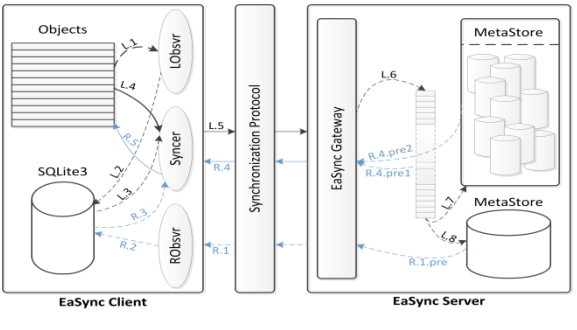

Fig. 2. Timestamp based synchronization protocol.

When the gateway receives an update request, L.6) it first starts conflict detection, and if no conflict happens, it then create an global timestamp by reading the server side clock, and updates the record in server side. In L.7), the server responses the EaSync client with the global timestamp, and the client record it in a local snapshot table. On the other side, R.1) The Robsvr periodically polls the server with a global timestamp, which is returned by the last polling, to fetch the change log on server to find which files are changed, and then R.2) inserts the change log batches into the to-be-synced pool, with setting the change type to be "changed by remote". At the meantime, the client will record the responded time as the next query time. Just like L.3) when the periodical Syncer daemon is executed, R.3) will be called. In this step, Syncer firstly query the to-be-synced pool to find the record sets where a remote synchronization is needed. Then Syncer literately runs R.4) and R.5). The former fetches the updated data from EaSync server and the later store them in local.

2.2 Consistency Model

It is common that the network is always partitioned. Also, in some situation, the client even can only access the data in its own device. So it is not possible to maintain all the replications on different machines in a strong data consistency. In EaSync we provide an eventual consistency[8] for the file synchronization. However, in the eventual consistency model, conflicts are unavoidable. To solve conflict problems, there are two key questions to be answered, and they are 1) when to discover the conflict, and 2) whose duty to resolve the conflict.

For the first question, EaSync detects the conflict at the synchronization phase by the client. Basically, in a partitioned network there are two conflict forms: RAW (Read-After-Write) in which a client reads the value of a data item that is being concurrently updated by a second client, and WAW (Write-After-Write) in which two clients update the same data item in incompatible ways.

For both of these two conflict types, we provide an optimistic strategy for conflicts detection: we accept all the user operations in the client and leave the conflict detection at the synchronization phase. When EaSync client starts to synchronize their data, it detects conflicts first. It firstly check whether there is a same file exists on the server side whose global update time is later than the last global update time stored in the client. If it is true, it means that some other clients have updated the data before this synchronization. In this situation, a WAW conflict is detected. RAW conflicts may happen in the eventual consistency schema too. For example, if client A updated some data, and before it is synchronized to the server, client B will not get the freshest data. This can also be resolved at the synchronization phase by either synchronizing the data in which client B does not update this file, or turning RAW conflict into WAW conflict in which client B updates the file. With this optimistic conflict detection strategy, EaSync can detect the conflicts at the synchronization phase.

After the conflicts are detected, we need to resolve these conflicts. As we analyzed in previous, the main conflict is the WAW conflict at synchronization phase. In detail, the problem is how to resolve the situation when there is a file whose update time is later than the last update time on the client. EaSync resolves these conflicts by creat-

ing a new version on the server side with a suffix .timestamp.clientname. After the EaSync client finishes its synchronization phase, the user will find the conflict versions. EaSync leaves it to the users, relying on humans effort for resolving conflicts.

3 Optimization and Implementation

In the file synchronization service like EaSync, user data is always revised frequently. There will be a bundle of data which is very similar. If we split the files into chunks, there will be a number of same chunks from different files. In EaSync, we use a deduplication method at the client side. EaSync uses deduplication together with S-RSync[9] and we name this method as DS-Dedup.

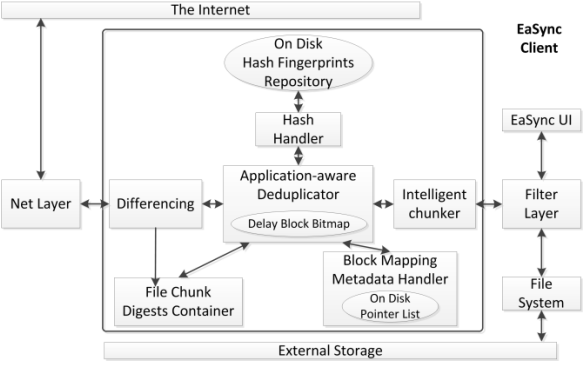

Fig. 3. Overview of DS-dedup.

Figure 3 shows the overview of DS-Dedup. When a file is modified, EaSync will send a request to the DS-Dedup module to synchronize the data to serve. DS-Dedup serves EaSync request by the Intelligent chunker. It splits the data into chunks intelligently according to the frequent of file modification. DS-Dedup uses content defined chunking (CDC) to split for the frequently updated files, and static chunking (SC) for the files not frequently updated. After the file data is split into chunks, the chunks enter into the AAplication-aware Deduplicator (AADedup)[2]. AADedup controls the deduplication of DS-Dedup. When a chunk arrives, it first calculates the fingerprint, and searches in the Hash Fingerprints Repository. It the fingerprint of a chunk exists, AADedup will generate a point to the chunk data and stores it in the On Disk pointer List. This is done by Block Mapping Metadata Handler. Otherwise, if there is no fingerprint for the new coming data chunk, AADedup will generate a new fingerprint by Hash Handler and add it to the Hash Fingerprints Repository. In order to synchronize data to the EaSync server side with S-Rsync differencing algorithm, an File Chunk Digests Container component is used in DS-Dedup to store the chunks information including the chunk finger print of the file. When S-Rsync start to check the differences, it reads the chunk fingerprint information and generate the differences with S-RSync algorithm. After that the difference is then transferred to the EaSync server side with S-RSync algorithm.

4 Evaluation

We set up an experimental platform with 2 PCs and 1 laptop as EaSync clients and Panasas ActiveStor cluster for Gateway, MetaData and DataStore server. The clients and server are connected by a local area network. With this experiment platform, we evaluate EaSync as follows: First, we evaluate EaSync by comparing with an open source project named with iFolder. Second, we have evaluated the EaSync in the aspect of synchronization protocol, in which we compare S-Rsync with Rsync. Finally, we evaluate the benefit and cost of deduplication.

4.1 Performance Comparison

In this section, we compare EaSync with iFolder for the operations latency of creating, updating, renaming and deleting.

We have collected a set of workload which is used and modified in our daily work. We select operations on files with different sizes and different types. The sizes are about 50KB, 100KB, 500KB, 1MB, 5MB, 10MB, 50MB, 100MB, 500MB, 1GB. We do not present the latencies for files smaller than 50KB and files larger than 1GB. The latencies of EaSync and iFolder for files smaller than 50KB are very close. Also, the files larger than 1GB are not common in our usage schema of file synchronization between different devices.

In the experiments, for creating operation, we put the files with different sizes into the folder which are monitored by EaSync and iFolder, then we get the latencies as the creating operation latency. For updating operation, we modify the files by appending 1KB data to the files, and then record the latencies of data synchronization, and make these latencies as the updating latency. Figure 4 shows the latencies of different operations.

As the result shows, EaSync outperforms iFolder for most of the operations including create, update, rename and delete. The main reason for the larger latencies for create and update is that iFolder uses RSync synchronization algorithm to synchronize the differences of the files. While RSync will read data from the server first, then compare the data with that in client side. However, EaSync uses our proposed SRSync algorithm to do the operations, it reduces the cost a lot, as shown in the figure. Another information included in the figure is that when the size of the file increases, the latencies of iFolder increases faster than that of our EaSync. In summary, compared to iFolder, EaSync outperforms in the operation latencies.

4.2 Effect of Ds-Dedup

EaSync uses deduplication to reduce the capacity and transmission cost. In this section we evaluate the effect of deduplication for EaSync. First, we evaluate the storage capacity used by a frequently revised file. We compare DS-Dedup with other strategies with different chunk size settings. As the deduplication method is dependent on the chunk size of file split. In our experiments, we uses static chunk split method with the chunk sizes of 0.5KB, 1.0KB, 1.5KB, and 2.0KB.

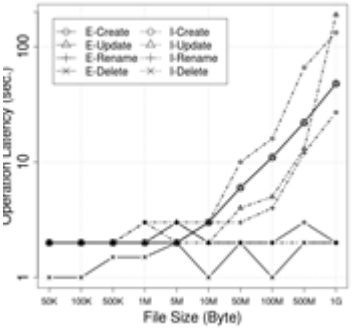

Fig. 2. Operation latencies.

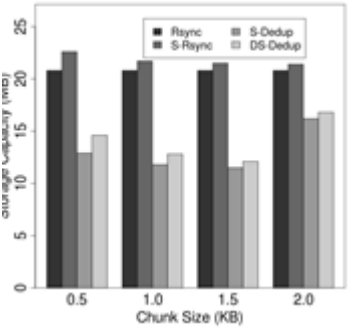

Fig. 3. Storage capacity used.

Figure 5 shows the result of storage capacity used under different strategies. As the result shown, Rsync uses about 20.8MB for the storage capacity. However, S-Rsync uses a little bit more. This is mainly because of that S-Rsync needs to store the chunk digest information for the files. While as S-dedupe uses deduplication method for both the data chunks and the metadata, it costs the least space. For the same reason as S-Rsync to Rsync, DS-Dedup uses a little more space than S-dedupe because of the space cost of chunks digest information.

From the result we can find that, comparing with the strategies with no deduplication methods, DS-Dedup and S-dedup is always better. DS-Dedup can reduce the storage cost to about half of that of Rsync and S-Rsync. Also, it can be found that the storage cost is dependent on chunk size. When deduplication is used, the chunk size should be selected carefully. In this experiment setting, 1.5KB is the best size for deduplication. Another benefit of client side deduplication comes from the transmission cost to synchronize the updated data to the server. As the redundancy of the data is deleted, EaSync only needs to transfer part of the whole data which is enough to reproduce the original data.

5 Related Work

EaSync shares several considerations with the other prior works. Services like Dropbox, UbuntuOne DBank, and Wukong[4] are probably the closest existing work to our EaSync. We share the overall architecture, which is client-server based. However, EaSync is designed not only for the file synchronization service, but also for research purpose, which is designed to be an open source project. While the existing services like Dropbox, UbuntuOne are always close source projects, especially the server side design where much more research issues and large scale data center challenges may be discovered. So EaSync shares these services with similar ideas, but contains more values for research community. EaSync also shares a lot of technologies and ideas with some synchronization methods, like Rsync[6,7], and the version control services. However, most of the traditional synchronizers are invoked explicitly by an action

from the user (issuing a synchronization command, clicking the synchronization button, etc.), while EaSync runs a background daemon and synchronize the updates automatically and transparently for the users. It definitely improves the user experience and eases the user's operations which may introduce unconscious errors.

Differencing algorithm is an important technology in file synchronization service. Recently, there are some differencing algorithms aiming at special application or special occasion. Rsync[7] is widely used for synchronization, backup and recovery system. And there are also some other algorithms adopting other consideration like, Delta-encoding. The work[5] uses "delta" vcdiff[3] encoding way to improve performance of HTTP traffic. The algorithm vcdiff is one of the best delta encoding algorithms. Every differencing algorithm has its own advantages on the reasonable size of difference or computing overhead, according to its special scene. Because of the simplicity and efficiency, Rsync is used widely in synchronization system.

6 Conclusion

We present EaSync, a transparent file synchronization service across multiple machines in this paper. EaSync characterizes itself with several unique features: First, it supports file synchronization in local area network environment. Second, it proposes several key technologies for file synchronization oriented service, including a dual-timestamp based synchronization protocol, DS-Dedup. Third, it explorers an open source implementation for research community.

Acknowledgments. The authors would like to thank the anonymous reviewers for their comments and kindly suggestions. This work is supported by the National Natural Science Foundation of China under Grant No. 60736013, Grant No. 61120106005, Grant No. 61025009 and Grant No. 60903040.

References

1. Dropbox. http://www.dropbox.com
2. Fu, Y., etc.: Aa-dedupe: An application-aware source deduplication approach for cloud backup services in the personal computing environment. In: Proceedings of the CLUSTER 2011.
3. Korn, D., etc.: The VCDIFF generic differencing and compression data format. 2002
4. Mao, H., etc.: Wukong: A cloud-oriented file service for mobile Internet devices. In: Journal of Parallel and Distributed Computing (2011)
5. Mogul, J., etc.: Potential benefits of delta encoding and data compression for http. In: Proceedings of ACM SIGCOMM 1997.
6. Rasch, D., etc.: In-place rsync: File synchronization for mobile and wireless devices. In: Proceedings of the USENIX ATC 2003.
7. Tridgell, A., etc.: The rsync algorithm. Australian Natl. Univ. Canberra, 1996.
8. Vogels, W.: Eventually consistent. Communications of the ACM 52(1), 40{44 (2009)
9. Zhang, H., etc.: S-rsync: An efficient differencing algorithm with locally chunk digests generating for file synchronization services. In: Proceedings of the Humancom 2011.