D-Sync: A Distributed, Automatically Synchronized File Hosting System

Paul Elliott
University of Oregon
paule@cs.uoregon.edu

David Stevens
University of Oregon
dstevens@cs.uoregon.edu

***Abstract*—File hosting systems such as Dropbox and Google Drive are widely used Internet tools for data storage and backup, collaborative work, and file sharing. These systems, while increasingly popular, rely on centralized storage of data, which pose inherent privacy concerns for certain types of users and working groups. In this work, we present D-Sync, a distributed file hosting architecture that allows users to store and share data from workspace directories on local machines. All workspace directories are updated and synchronized whenever an offline user connects to the system, and whenever an online user makes a change to his workspace directory. [FILL IN after evaluation]**

I. Introduction

Of the wide range of today's Internet tools, file hosting systems such as Dropbox and Google Drive are certainly among the most popular. These systems offer convenient (and often cheap) means to store and backup data, in addition to making that data available to any device with Internet access and the proper credentials. The ever-growing ubiquity of Internet access and the advent of cloud storage have helped these systems flourish, allowing them to serve billions of files for millions of users.

As the most popular file hosting system on today's Internet, Dropbox leads the way in this growing domain. Since January 2010, Dropbox has steadily expanded from about 4 million to 100 million users, and increased its usage rate from roughly 250 million to 1 billion files saved per day [3]. Stop for a moment and consider the sheer magnitude of these statistics: even roughly averaged, this means that Dropbox users collectively save over 1 million files *every two minutes*.

This frequency of usage points to a key aspect of Dropbox's popularity: its capacity for collaborative work. The system allows its users to share data with other select users in the so-called working group, and also keep a local copy of the shared data on any owned machine or device. Most importantly, Dropbox synchronizes quickly and automatically across local and cloud replicas by employing **delta encoding**, in which only the *changes* to files are saved and uploaded. The efficiency of this synchronization strategy makes Dropbox a convenient tool for both synchronous and asynchronous collaboration among working groups. The benefits of these features have not gone unnoticed, and continue to attract attention [1], [5].

When we consider data privacy in Dropbox and other cloud-based file hosting systems, however, we see that not everything is silver lining. Privacy issues with cloud services have recently fallen under scrutiny by the security community [16], [15], and Dropbox has not escaped this critical eye. Aside from a few exposed security issues [11], [12], the top file hosting system has been criticized for its policies on data discretion. In order to facilitate delta encoding, Dropbox maintains encryption keys to calculate hashes for saved files. These keys enable both Dropbox and warranted law enforcement agencies access to user data [4]. While this capacity also improves the system's throughput, it bears too high of a privacy cost for many types of working groups.

One solution to the privacy issues of centralized data would be to distribute data storage across users' local machines. The file hosting system itself would remain blind to this data, and merely marshall encrypted updates between user machines to handle synchronization. For such a distributed file hosting system to be successful, it would have to achieve the following design goals (in addition to privacy):

- **Accuracy:** The system must be able to properly synchronize all updates across all *online* users. In other words, whenever a user needs to pull data from the system, it must pull a completely up-to-date copy. Furthermore, a user should not push outdated data to the system.
- **Transparency:** The system itself should largely be transparent to the user. To facilitate transparency, the system must enable automated and asynchronous updates for shared data. When a user is online, changes should be pushed immediately after they occur. In addition, a user should not have to block and wait for the system to update.
- **Efficiency:** The system must be efficient with respect to the latency of message and data transfers. In particular, the amount of extra information (beyond file data) that is passed between networked entities should be small.

In this work, we present D-Sync, a distributed and automatically synchronizing file hosting architecture that addresses the privacy issues of centralized data storage. D-Sync addresses these concerns by distributing data across user machines, called **clients**. [FILL IN high level details of how this would work] [FILL IN discussion of broker]

[FILL IN issues that need to be addressed]

[FILL IN high level details of how the prototype system handles these]

[FILL IN evaluation details: how, main takeaways]

The contributions of this work include:

- *D-Sync: a distributed file hosting architecture.* automated synchronization of files and protects the privacy of its

users. Our architecture includes schematics for its major components–clients and brokers–as well as a communication protocol and automated synchronization strategy. While similar proposals exist, to the best of our knowledge our work is the first to posit such an architecture as a privacy-preserving alternative to conventional file hosting systems.
- *An evaluation of distributed file hosting architectures.* In evaluating our prototype system's performance and accuracy, we provide a first look at how such architectures may perform on various network scenarios. Furthermore, we conduct a comparative analysis of Dropbox's performance, and present the results in conjunction with our prototype evaluation.

The remainder of our paper is organized as follows. In Section II we highlight previous work related to our project. Section III covers our system design, including its major components and design decisions, and Section IV details the prototype we built using this design. We evaluate our system in Section V, specifically analyzing its performance and synchronization correctness. In Section VI we discuss our work and future directions, and finally we conclude in Section VII. Section VIII contains a brief description of our code and division of labor on this project.

II. Related Work

The topics of synchronization and replication strategies have been considerably researched within the general domain of distributed file systems. Much of this work has focused on replica maintenance for the purpose of data availability [14], [6], [8], [2], [7]. Pu et al's work discusses various replica control mechanisms for maintaining epsilon-copy serializability (ESR), a correctness criterion that allows asynchronous maintenance of mutual consistency for replicas [14]. However, their evaluation does not extend to specific domains of distributed file systems. Their research also lays valuable foundations for evaluating replication strategies in specific domains (i.e., [7]). While we also build on Pu et al's groundwork, our emphasis on automated synchronization and privacy constitute a distinct focus from other follow up research.

In another similar work, Chun et al evaluate replication strategies for storage systems distributed over the Internet [2]. Their research focuses on data durability, the assurance that data put into the system is not lost due to disk failures, and claims that this property be held more important than conventional availability. While their work uncovers a valuable property that should be widely applicable in distributed storage systems, our research focuses specifically on availability as a necessary prerequisite of automated synchronization.

Lastly, a couple works on distributed collaboration systems closely mirror that of our own [13], [10]. These works discuss peer-to-peer based architectures for distributed collaboration tools, including proposed system designs built atop existing architectures (GitHub) and provably correct synchronization models. [FILL IN why different]

III. System Design

Here we cover the design of our system architecture. We first state the overall scope we used to work toward our design goals of accuracy, transparency, and efficiency. We then describe the main components of our system–namely, the client and broker programs, followed by the communication protocol that these entities use. In these sections we will highlight specific design decisions that we needed to address in order to achieve our stated goals, and explain how we handled them. We finish this section with a [NECESSARY?].

A. Scope

Before we could begin designing a distributed file hosting architecture, it was necessary to define the scope for our work. Our assumptions for this work are as follows:

- *Connection reliability:* The individual components of our system connect via TCP stream sockets in order to pass messages and data. This precludes reliable transmission of data, with failure detection and subsequent retransmission. While TCP handshakes will increase the latency of message passing, reliable transmission allows us to focus on the accuracy of our system. [FILL IN possibly point to future work in unreliable transmission?]
- *Conflict resolution:* Automated synchronization necessarily implies instances of conflicts between closely committed updates. Previous work ([**?**], [9]) has been done on conflict resolution strategies in distributed domains, generating several viable solutions. We rely on these promising schemes in place of proposing our own solution, which is beyond the scope of this work.
- *Security:* Our work falls in the domain of networked applications, and therefore is subject to a litany of network security attacks. The security community has addressed such threats in a body of literature that is too vast to properly reference. Instead we indicate where we leverage secure hashing algorithms (SHA) and public key infrastructure (PKI) to facilitate the integrity and privacy of our system, and leave the rest to the capable hands of security researchers.

B. D-Sync Architecture

[FILL IN high level description of system and sync model]

1) Client:

2) Broker:

C. Communication Protocol

IV. Prototype System

V. Evaluation

We evaluated our system with respect to performance and accuracy. For the former, we measure the latency of key synchronization tasks across different network scenarios and compare the results with Dropbox. For the latter, we test specific use cases where synchronization could fail and demonstrate that the system properly handles these cases.

A. Performance

In order for D-Sync to be a feasible architecture for distributed file hosting, it needs to perform comparably to existing systems. We therefore evaluate our prototype system with respect to the latency of updates. We not only show that D-Sync scales reasonably well, but furthermore outperforms Dropbox in certain cases.

1) Metrics and Parameters: In order to evaluate the latency of our system, we measure the time it takes for adding a single file to propagate across all online clients. We focus on adding files since these updates will necessarily have the greatest latency, given that the system must propagate raw file data to all online clients. We measure the total latency as: (time stamp of when the last client receives the entire file) - (time stamp of when the sending client starts sending the file).

We define the parameters of our performance tests in the following manner. First, we evaluate two separate network scenarios: a LAN scenario with two computers networked over the UO Secure wireless network, and a WAN scenario with the same two computers on two separate home wireless networks. Within these two testing scenarios, we test additional parameters to help determine how our architecture scales. First, we evaluate working group sizes of 2, 4, and 8 clients. Should the system experience more than linear growth in latency as the working group size increases, then our system will not reasonably scale. Finally, we evaluate our system with file update sizes ranging from 2^n MB, where $0 \leq n \leq 10$. We do this in order to gain insight on how our system scales with respect to file size.

2) Setup: With our metrics and parameters defined, we proceed with our tests in the following manner. In either network scenario, we designate one computer to run the broker program and the other to run the variable number of client programs. Though the client programs all run on the same computer, round trip time is still necessary since all communication between clients must pass through the broker.

To conduct a test we start the broker program, followed by the client programs. Once all clients are connected to the broker, we move a text file of the appropriate size into the workspace directory of a given client, and measure how long it takes for all other clients to receive the file. To ensure that our measurements are correct, we synchronize the local clocks of both computers using the network time protocol. We repeat the sending of text files in a serial fashion until all files have been sent.

In the following section, we cover the results of these tests.

3) Update Latency:

4) Comparison to Dropbox:

B. Accuracy

In addition to providing an efficient solution with low latency, it is important that our architecture provide data consistency across different scenarios. In other words, the system must properly synchronize all updates across all online users. In this section, we evaluate specific scenarios where where the replicas in a given working group could become inconsistent across clients, and demonstrate how our system handles these scenarios.

1) Client Departures: A client could depart from the system in several ways, the most common of which are going offline and crashing. There are three primary scenarios that could cause data inconsistency in the event of a client departure:

(1) the departed client could make offline changes to his local workspace;
(2) one or more online clients could make changes while the departed client is offline; or
(3) both the departed client and one or more online clients could commit conflicting changes.

In all of these cases, the system would have to properly update shared data when the departed client returns and does an initial batch update.

We preliminarily evaluate these scenarios in stress tests using eight clients and one broker in a LAN setting. We start each client and connect them to the broker, and begin committing changes on various local workspaces. At random intervals, we take one or more clients offline and then bring those departed clients back after a fixed length of time (2 minutes). Throughout the test we continue to commit changes to the local workspaces of all clients, online or offline.

Our results from four such tests show that the initial batch update properly handles all cases of potential data inconsistency. Across four tests, we observe 31 instances of case (1), 26 instances of case (2), and 12 instances of case (3). In case (1), the departed client's offline changes are immediately pushed to the broker, who commits them to the rest of the system. Case (2) is similarly handled; the broker pulls up-to-date files from the nearest online client and pushes them to the returning client. For case (3), our prototype broker accepts the first offline change it receives, discarding subsequent updates with the same timestamp. While a better solution would involve merging or handling such conflicts in some manner, conflict resolution strategies are beyond the scope of this work.

Furthermore, we note that our system maintains accuracy even when other clients depart during a re-entering client's batch update. In these cases, the second departing client will also be updated upon return. In this manner, all online clients will maintain a current workspace for all online updates. These preliminary tests show that the D-Sync architecture keeps data consistent in the presence of arbitrary client departures.

VI. Discussion

FOLLOW UP FROM OUR WORK?? (more rigorous testing of accuracy) Fully distributed broker Other synch strategies (for performance) security-¿evaluate performance hit

VII. Conclusion

References

[1] Mercurial on dropbox.

[2] B. Chun, F. Dabek, A. Haeberlen, E. Sit, H. Weatherspoon, M. Kaashoek, J. Kubiatowicz, and R. Morris. Efficient replica maintenance for distributed storage systems. In *Proc. of NSDI*, volume 6, pages 225–264, 2006.

[3] J. Constine. Dropbox is now the data fabric tying together devices for 100M registered users who save 1B files a day, 2012.
[4] J. Constine. How dropbox sacrifices user privacy for cost savings, 2012.
[5] A. Dachis. How to use dropbox as a killer collaborative work tool, 2011.
[6] O. Damani, A. Tarafdar, and V. Garg. Optimistic recovery in multi-threaded distributed systems. In *Reliable Distributed Systems, 1999. Proceedings of the 18th IEEE Symposium on*, pages 234–243. IEEE, 1999.
[7] D. Ford, F. Labelle, F. Popovici, M. Stokely, V. Truong, L. Barroso, C. Grimes, and S. Quinlan. Availability in globally distributed storage systems. In *Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation*, 2010.
[8] S. Goel and R. Buyya. Data replication strategies in wide area distributed systems. Technical report, ISBN 1-599044181-2, Idea Group Inc., Hershey, PA, USA, 2006.
[9] W. Hurley and K. Habermehl. Collaborative model for software systems with synchronization submodel with merge feature, automatic conflict resolution and isolation of potential changes for reuse, Jan. 13 2004. US Patent 6,678,882.
[10] F. Merle, A. Bénel, G. Doyen, and D. Gaïti. Decentralized documents authoring system for decentralized teamwork: matching architecture with organizational structure. In *Proceedings of the 17th ACM international conference on Supporting group work*, pages 117–120. ACM, 2012.
[11] M. Mulazzani, S. Schrittwieser, M. Leithner, M. Huber, and E. Weippl. Dark clouds on the horizon: Using cloud storage as attack vector and online slack space. In *USENIX Security*, volume 8, 2011.
[12] D. Newton. Dropbox authentication: insecure by design, 2011.
[13] G. Oster, P. Urso, P. Molli, and A. Imine. Data consistency for p2p collaborative editing. In *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, pages 259–268. ACM, 2006.
[14] C. Pu and A. Leff. *Replica control in distributed systems: as asynchronous approach*, volume 20. ACM, 1991.
[15] C. Soghoian. Caught in the cloud: Privacy, encryption, and government back doors in the web 2.0 era. *J. on Telecomm. & High Tech. L.*, 8:359, 2010.
[16] M. Van Dijk and A. Juels. On the impossibility of cryptography alone for privacy-preserving cloud computing. In *Proceedings of the 5th USENIX conference on Hot topics in security*, pages 1–8. USENIX Association, 2010.

VIII. Appendix