CON$: a Concurrency Control Mechanism for Dropbox

John Dinh
Swarthmore College
500 College Avenue
Swarthmore, PA 19081
jdinh1@swarthmore.edu

Ivana Ng
Swarthmore College
500 College Avenue
Swarthmore, PA 19081
ing1@swarthmore.edu

Abstract

Dropbox, a cloud storage service, currently does not provide a concurrency control protocol. This means multiple users concurrently editing a file stored on a shared Dropbox directory may overwrite each other's changes. We present CONS, a simple concurrency control mechanism that utilizes Dropbox's syncing feature to implement locks on files stored on Dropbox's servers. CONS is also packaged with a GUI text editor, through which text-based files are viewed and modified. We evaluate our mechanism by testing various scenarios of how users may interact with CONS, and measuring the latency associated.

1 Introduction

Dropbox is a cloud storage service that allows you to store photos, documents, music and other files. You can upload files directly to the Dropbox website. You can also install the Dropbox desktop application onto your personal computer, which will then set up a folder on your local hard drive called "Dropbox." Any changes made to this folder (e.g. add/remove files, modify existing files) will also be reflected in your account on the Dropbox website. This means that you can access files anywhere, on any machine. Besides offering a cloud-based backup system, Dropbox also allows you to share directories with other users. Shared directories, and any changes made to their contents, are synchronized across all users.

The current version of Dropbox does not prevent users from overwriting each other's changes. In other words, it does not have a concurrency control protocol. Consider the following situation. Two users, Annie and Britta, are editing a text file called `community.txt` at the same time. This file is located in a Dropbox directory that they share, and they are each editing a local copy of the file. If Annie or Britta is using a computer that has the Dropbox application installed, then the file is already located on the local hard drive. Otherwise, it can be downloaded from the Dropbox website.

After a user modifies the file, she must synchronize ("sync") so that the changes are reflected across all servers. If she is on a computer that has the Dropbox application installed, she synchronizes simply by saving the file. However, if she had downloaded the file from the website, she must manually upload the file to the Dropbox website.

Now, suppose Annie finishes modifying `community.txt` and syncs her changes to the shared folder. Next, Britta saves the changes she has made to her local copy of the file, and syncs it back to Dropbox. As a result, Britta's copy becomes the "official" version, and Annie's changes are overwritten.

This lack of concurrency control essentially relegates Dropbox to being a basic backup service. However, we see great potential for Dropbox to become a real-time collaboration tool, much like Google Docs. We believe Dropbox can be a useful service for multiple users, such as within companies or among college students, to work on files concurrently. Currently, however, Dropbox does not ensure data consistency or provide multi-version control for this type of user activity.

In this paper, we present CONS, a simple concurrency control mechanism that utilizes Dropbox's syncing feature to implement locks on files stored on Dropbox's servers. When a user wants to access a file located in a shared Dropbox directory, he applies for a lease, or temporary lock, so that he is the only user who has permission to open the file.

In addition to creating a lightweight and robust mechanism for concurrency control, we also implement a GUI Text Editor, called the CONS Text Editor, that works in concert with CONS. Files that end in `.txt` are opened in this editor, which automatically syncs changes to the Dropbox server. For all other file types, the user must manually open it in an appropriate application and sync her changes back to the website as well.

The rest of this paper describes the implementation and evaluation of CONS in detail. In Section 2, we discuss related works such as the concept of leasing and consensus protocols like Paxos. In Section 3, we describe the Dropbox application programming interface, and how CONS interacts with it. In Section 4, we describe the implementation and algorithm of CONS. In Section 5, we evaluate CONS by examining latency issues and the ownership of leases, both of which are key to CONS's performance. Finally, in Section 6, we describe the implications of a concurrency control protocol like CONS for Dropbox, and we suggest areas for future work.

2 Related Works

CONS is based on two concepts: leasing and consensus. Leases give users the permission to read and/or write to a file exclusively. The concept of a lease was first introduced in 1989. Gray and Cheriton proposed this time-based mechanism as a means of maintaining cache consistency in distributed file systems (Gray and Cheriton, 1989). When the server assigns a lease to a client, it guarantees that the server won't update the file associated with the lease without the permission of the client (Kon and Mandel, 1995).

A delayed-write lease allows a client to write to its own cached data and then update the server asynchronously. Such a lease can only be assigned to one client at a time. Before the server can assign a delayed-write lease, it must ensure that no other client has a lease for the same file (Kon and Mandel, 1995).

CONS associates each file in a Dropbox directory with a delayed-write lease. At any given moment, this lease can be assigned to exactly one user. Once a user has the lease, he has an exclusive-write lock on the associated file. As long as the lease has not expired, other users cannot modify the associated file. If the lease does expire, other users may apply for permission to transfer the lease.

According to Kon and Mandel, the performance of a lease-based system is largely dependent on the validation period of a lease. If leases last only a short period of time, the server may have to constantly poll the user and ask if it would like to renew its lease. On the other hand, if leases have a long validity period, other users must wait a long time before they can apply for the lease, and user crashes and network partitions may not be handled efficiently (Kon and Mandel, 1995). Finding the optimal expiration period of a lease is crucial for CONS, since it is an application that allows for time-sensitive group collaborations on Dropbox.

The other component of CONS is consensus. This is the process of agreeing on one result among a group of participants. If multiple users try to access a file in Dropbox at the same time, they must all be able to reach a conclusion as to who ultimately has permission to open the file.

The Paxos protocol (Lamport, 2002) involves a coordinator who collects proposals and determines whether a majority (consensus) has been reached. Similarly, CONS uses a centralized authority to solve its consensus-like problem. The Dropbox server acts as the de-facto coordinator by determining which user's application for a lease is accepted, in the event that multiple users concurrently ask for a lease on a file. By interfacing with CONS and the Dropbox website server, users are able to communicate with each other indirectly.

3 Using the Dropbox API

CONS interacts with Dropbox files through Dropbox's application programming interface (API). We decided to use the API instead of the desktop application, because it gives us more control over our application. We can detect when interactions with Dropbox are successful, and we can check when interactions are finished. This allows us to create a more robust application while handling data. If CONS detects that a sync was unsuccessful, it will notify the user and save a copy on her local disk drive.

Another advantage for using the Dropbox API over the Dropbox desktop application is that users can run CONS on any computer. There is no need to install anything in order to use it. This is particularly useful when a user is working in an environment where she does not have system administrator permission to install and use the Dropbox desktop application.

CONS uses five methods from the Dropbox API: `get_file`, `put_file`, `remove`, `account_info`, and `metadata`. We describe each of these below.

- `get_file(path)` This method takes the full Dropbox path to the desired file as a parameter and returns the file's data.

- `put_file(path, file_object, overwrite)` This method takes as parameters the full Dropbox path to the destination directory, the file object, and a boolean value for overwrite. If the overwrite flag is True, Dropbox will overwrite any pre-existing file with the same name. Otherwise, Dropbox will modify the filename of the uploaded file in the case that the original filename already exists within the directory. `put_file` returns a dictionary of metadata associated with the newly uploaded file. We use `put_file` to upload lockfiles and to upload a file when a user has saved his or her changes.

- `file_delete(path)` This method takes as a parameter the path to the file to be removed, and returns a dictionary of metadata associated with the deleted file. We use `file_delete` to remove locks when the user does not need them or when leases expire.

- `account_info()` This method returns a dictionary with the user's account information. We use this to extract the unique user identification number, which we then use in the lock file in order to determine the owner of that lease.

- `metadata(path)` This method takes a path to a file, and returns a dictionary of its associated metadata. We extract the timestamp from a successful lock file and use that to mark the beginning of a user's lease.

The Dropbox Software Development Kit provides a sample client script called **cli_client.py**, which we adapted into the immplementation of CONS. It uses a command line interface to link to and navigate a user's Dropbox account.

4 Implementing CONS

Files stored on Dropbox are synced across all platforms, including computers, phones and the Dropbox website. Dropbox is also able to prevent multiple users from uploading files with the same name to a shared directory, if the file's overwrite flag is set to False. That is, if multiple users upload <filename>to the same folder, Dropbox randomly chooses one version to take precedence over the others and renames the other ones as <filename>(1), <filename>(2), and so on. CONS exploits this property to manage lock assignment.

4.1 Applying for a lease

CONS has three phases: authorization, application and verification. The first phase is a simple one-time, log-in process. First, the user - we will call her Britta - types `login` into the command line. CONS then prompts her to type in her email address and password (i.e. her login information for Dropbox.com). CONS then interfaces with the Dropbox API to request, authorize and store an authentication token.

Britta proceeds to the directory that contains the file she wishes to modify. She may use Unix commands such as `ls`, `cd` and `pwd` to navigate the file system. She types `open community.txt` in order to get the file that she wants.

CONS then checks the directory to see if an associated lockfile already exists. A lockfile is any file whose name contains the string `.lock.` at the front of the filename. The file itself contains three pieces of information: a username, a user ID, and a timestamp. In this case, CONS searches the directory for `.lock.community.txt`.

If the lockfile already exists, CONS proceeds to download it and read its contents. It checks the timestamp and calculates when it expires. We will discuss lease validity and expiration in Section 4.3. If the lease has not expired, then CONS returns a message to Britta, notifying her that another user is accessing the file. If the lease has expired, CONS applies for a lease to transfer the lockfile. This process is discussed in Section 4.4.

If the lockfile does not exist, CONS calls `put_file(.lock.community.txt)` to upload a lockfile with Britta's name and user ID written inside, to the Dropbox directory in which `community.txt` is located. This function returns a dictionary of metadata associated with the lockfile.

Here, it is important to note that Dropbox has a duplication protocol, which we utilize during the application process. If multiple users try to access <`filename`>, each of their instances of CONS calls `put_file(`<`filename`>`)`, with the overwrite flag set to False. This means that each version of <`filename`> will be synced to the directory. Dropbox, as the central coordinator, decides which one is the "original" copy and appends version numbers to the other files. Thus, the directory would contain a <`filename`>, <`filename`>`(1)`, <`filename`>`(2)` and so on. The API call `put` returns metadata pertaining to the newly named file.

In Britta's case, CONS calls `get_file(.lock.community.txt *)`, where * may be a version number. This may occur if other users have tried to access `community.txt` at the same time as Britta. The `get` call downloads a local copy of the file it had just uploaded, and then writes the timestamp, which was extracted from the metadata, to the lockfile. Next, CONS calls

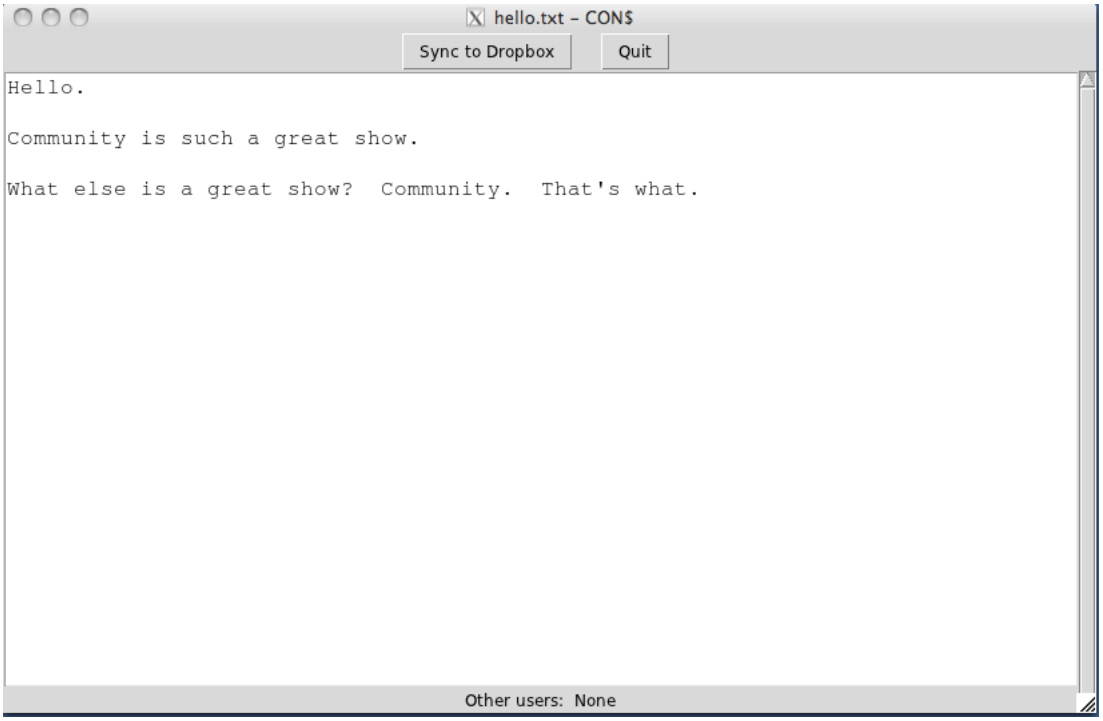

Figure 1: The CONS Text Editor.

`put_file(.lock.community.txt *)` to sync the modified lockfile to the Dropbox server.

In the next phase, CONS verifies whether Britta's application for a lease was successful. CONS calls `get_file(.lock.community.txt)` and reads the file's contents. If it contains Britta's user ID, then the lease has successfully been assigned to Britta. Notice that the application and verification procedure requires a total of two `put_file` calls and two `get_file` calls.

Once CONS determines that Britta does have the lease, it calls `get_file community.txt` to download a copy of the file to the local hard drive. The file is then opened in the CONS Text Editor. We used Tkinter, a GuiProgramming toolkit for Python, to construct the GUI text editor. It can be used to view and edit non-proprietary text files, such as `.txt`, `.c`, and `.py` files. For all other file types, CONS prompts the user to manually open the file using an appropriate application.

Our text editor contains a button called "Sync to Dropbox." This is equivalent to the "Save" command in other editors. Whenever the user clicks this button, CONS syncs the changes made to that file back to Dropbox. Figure 1 is a screenshot of the CONS Text Editor.

4.2 Concurrency Control

If multiple users, say Annie and Britta, concurrently apply for a lease to the same file, i.e. `community.txt`, their respective CONS applications will go through the application process as described above. This means multiple copies of `.lock.community.txt` will be uploaded to the shared directory. The directory thus contains `.lock.community.txt` and `.lock.community.txt (1)`. In the verification process, however, their CONS applications download the "master" lockfile, `.lock.community.txt`, to see if it contains the respective user's ID.

4.3 Lease validity

The user has no control over the period of validity of his lease. A lease's validity period is only a few seconds long, but it is CONS, not the user, who renews it. This allows for the user to interact with files seamlessly, without consciously interacting with CONS's leasing system.

The lease remains active as long as the text editor is open, but this can be a problem if the user with the lease becomes idle. If Annie, for example, leaves her desk for an extended period of time and does not close the file, she would hold the lease indefinitely and no other user would be able to access that file. To remedy this problem, CONS keeps a timer that

runs for up to 15 minutes (this amount can be modified, according to users' needs). The clock is reset each time CONS detects cursor movement. If it does reach 15 minutes, CONS will automatically save the file and close the editor. However, CONS does not sync the file to the Dropbox directory, because we cannot be sure that Annie wants her changes to be synced across all users. This process ensures that other users can access the lease if the current owner of the lease becomes inactive.

In both cases, whether the user manually quits the CONS Text Editor or CONS force-quits due to user inactivity, the lease is subsequently released by deleting the file's associated lockfile. When the user actively releases ownership of the lease, CONS syncs the modified file back to the Dropbox directory, deletes the associated lockfile and deletes the local hard drive copy of the file.

4.4 Transferring the lease

While the lease for a file is still valid, other users can still apply for the lease. Say, for example, that Britta currently holds the lease to `community.txt`, and Annie would like to access the file. The first step in the application process, as mentioned earlier, is for Annie's CONS to check whether a lockfile already exists. If it does, CONS downloads a local copy of the lockfile and checks to see whether the lease is still valid. If the existing lease has expired, CONS applies to "transfer the lease." Essentially, CONS is applying for permission to delete the expired lockfile and then put up a new lockfile.

To do this, CONS uploads `.lock.lock.community.txt` to the directory. CONS then goes through the verification process by downloading a local copy of `.lock.lock.community.txt` and checking to see whether the file contains Annie's user ID. If it does, this means Annie's CONS has permission to delete the expired lockfile. If not, CONS's application to "transfer the lease" is denied, and CONS returns a message to Annie, notifying her that she cannot access `community.txt`.

CONS' application to transfer the lease can be denied, if multiple users concurrently apply for the same lease. For example, if both Annie and another user named Troy try to open `community.txt` and their versions of CONS concurrently apply to transfer an expired lease, Dropbox may sync Annie's lockfile as a different "version" called `.lock.lock.community.txt (1)`, rather than as the original `.lock.lock.community.txt`.

Note that obtaining permission to delete an expired lockfile does not automatically transfer the lease to that user. It simply prevents deadlock situations from arising, by allowing some user to delete the expired lockfile. Once the expired lockfile is removed, all users have the opportunity to apply for a lease to the desired file by going through the application and verification process explained in the previous section.

We return to Annie and Troy's situation to illustrate what may occur when multiple users are trying to access a file whose lease has expired. Let us assume that both Annie and Troy's versions of CONS applied to transfer the lease. So the Dropbox directory now contains `.lock.lock.community.txt` and `.lock.lock.community.txt (1)`. Both CONS applications download a local copy of the original, non-numbered version and read its contents. The file contains Annie's user ID, so her version of CONS has received the permission to delete the expired lockfile.

Now, Annie's CONS deletes the expired lockfile. The next step for CONS is to apply for a lease to open `community.txt`. But in the next time step, Annie's CONS may lag slightly, allowing Troy's CONS uploads a lockfile on Troy's behalf. Next, Annie's CONS also uploads a lockfile. In the verification phase, both instances of CONS find that the lockfile contains Troy's information. Therefore, even though Annie's CONS had received permission to transfer the lease, Troy's CONS actually received the lease and thus Troy ultimately opens `community.txt`. Though this may occur in some cases, it remains a seamless process for the user. The user simply types `open <filename>` into the command line, and the CONS Text Editor either opens or a message indicating that another user is accessing the file is printed to the screen.

The protocol for transferring a lease comes into play when users and/or the Dropbox server crashes or experiences network failures. When this occurs, CONS may not have the opportunity to delete lockfiles right away. As a result, when the user and/or Dropbox server comes back online, the Dropbox directory may contain lockfiles that have expired. In this case, users who apply for a lease would first apply for permission to "transfer the lease."

5 Evaluation

To evaluate CONS, we focus on two performance measures. First, we look at how CONS handles the ownership of a lease as well as the transferring of a lease from one user to another. Second, we examine latency between the time at which a user applies for a lease and the time at which he gains or is denied access to the file.

Specifically, we examine the following questions.

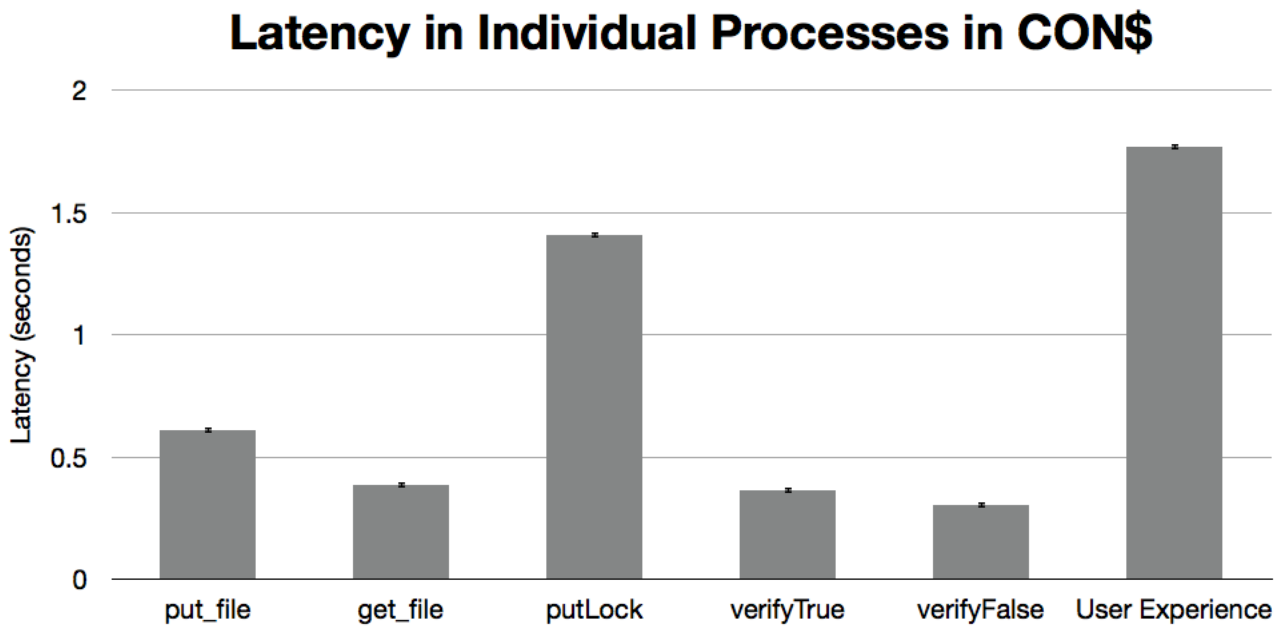

Figure 2: The greatest latency occurs when CONS calls `putLock`.

When a user attempts to access a file, how long does it take for CONS to open it in the text editor? After a user saves and closes the file (i.e. the lease is released), and another user tries to access the file, how long does it take for CONS to transfer the lease?

5.1 Ownership of a lease

CONS successfully implements delayed-write leases. This type of lease is assigned to one client at a time, and allows the client to write to its own cached data and then update the server asynchronously. While a user is holding a valid lease on a file, no other user may modify that file.

The transferring of a lease works similarly. When a user has the permission to transfer a lease, he is actually only allowed to delete the expired lockfile - and he is the only one who is allowed to do so.

5.2 Latency

Users must be able to access files quickly and obtain leases in a reasonable time. This can be an issue especially when multiple users are working concurrently on the same files. We evaluate the performance of CONS by examining latency in the individual processes that CONS performs when it applies for a lease.

As we discussed in Section 4.1, when CONS applies for a lease, it calls both `put <.lock.filename>` and `get <lock.filename>` twice. We believe this contributes significantly to the time between a user calling `open <filename>` and the CONS Text Editor opening the file.

To test this hypothesis, we recorded the time it took for CONS to complete the following functions:

- `put_file` - This function uploads a file and returns a dictionary of metadata.
- `get_file` - This function downloads a file.
- `putLock` - CONS contains a function called `putLock`, which is where the application process, described in Section 4.1, occurs.
- `verifyTrue` - After CONS applies for a lease, it verifies whether it received it by downloading the lockfile and reading its contents. In this case, CONS did receive a lock.
- `verifyFalse` - Similar to verifyTrue but we mimic a scenario with multiple users by placing a dummy lock file compose in order for `verifyLock` to return False.
- `User Experience` - Sum of `putLock` and `verifyTrue`

Our test case is a single user attempting to access a 1MB text file. We ran 100 trials each on a lab computer, Cinnamon, and recorded the latency for each

of the functions described above. Figure 2 shows our results. The last column, "User Experience," represents the time that it takes for the CONS Text Editor to open the text file that the user requests, once she calls `open <filename>`. As you can see, the greatest latency occurs when CONS calls `putLock`. This makes sense since this function requires four API calls (of `put` and `get`).

6 Conclusion

In this paper, we have implemented a lock-based concurrency control mechanism and GUI Text Editor for Dropbox. CONS allows users to gain exclusive-write locks on files in Dropbox directories. This prevents users from overwriting other users' changes. Concurrency control and data consistency is achieved without having users communicate directly with each other. The CONS Text Editor allows the user to view and edit text files, and it seamlessly syncs changes back to the Dropbox server.

There are several areas for future work. Currently, CONS assigns whole, exclusive-write locks. This means that once a user has a lease on a file, no other user may read or write to the file at all. This would certainly restrain productivity in a group collaboration environment. Lock granularity should be implemented, so that multiple users can concurrently modify the same file. Different types of leases and locks should also be implemented, since users access files in various ways. For example, a user who simply wants to read the file should receive a read-only lock, not an exclusive-write one. Introducing different types of locks and more granularity would make CONS a much more powerful tool.

References

C. Gray and D. Cheriton. 1989. Leases: an efficient fault-tolerant mechanism for distributed file cache consistency. In *Proceedings of the twelfth ACM symposium on Operating systems principles*, SOSP '89, pages 202–210, New York, NY, USA. ACM.

Fabio Kon and Arnaldo Mandel. 1995. Soda: A lease-based consistent distributed file system. In *In Proceedings of the 13th Brazilian Symposium on Computer Networks.*

Leslie Lamport. 2002. Paxos made simple. *SIGACT News*, 32:51–58.