# Dropbox Client

ASOIS Project

Necula Cristina

Faculty of Automatic Control and Computer Science

Advanced Software Services

## Problem Definition

Cloud services are becoming more and more popular as they provide a way to offload local computers computational resources usage. By offering file hosting services, cloud computing attracts a big number of users, as one user can access files stored on a remote server from almost any device that has an active internet conection.

Dropbox is a file hosting service that offers cloud storage, file synchronization, personal cloud and client software. It is available for a large series of opperating systems: Windows, Mac OS, Linux Based, Android, iOS, Blackbery and Windows phone. Dropbox provides an application that, when downloaded and registered, creates a folder inside the device file system that is synchronized on all the devices that use the same account to login.

This project aims to measure latency for

## Design

The proposed software solution for solving the problem stated in the previous chapter, we implement a Dropbox Client Application. Client Application has the following features:

1. Generate files of various dimensions
2. Upload content of given directory to Dropbox
3. Download content of given directory from Dropbox to local machine
4. Measure timing results for upload and download and corresponding file size

The client creates a new directory and generates an arbitrary number of files with different dimensions. The created directory content is covered and all the files are uploaded to Dropbox, measuring file size and the time needed for the actual upload. After all the files are uploaded, client application downloads them, also measuring every file size and time needed for download. All writing operations have to be sequential as we want to measure the latency of those operations.

For the purpose of this project, file sizes are to vary and the folder should not contain two files with same dimension. In order to be able to compare time needed for upload and download, same files that are uploaded to Dropbox have to be downloaded afterwards.

Dropbox has a special section for developers that want to include cloud sharing in their applications. In order to accomplish this feature, an SDK is provided under an open source license. The SDK is available for download for different programming languages: Swift, Python, .NET, Java, JavaScript, PHP and Ruby. To accomplish what we have proposed, Dropbox SDK has to be included in the client application.

## Implementation and Experimentation

In the implementation and experimentation phase, the design, or some aspect of the design is implemented to demonstrate feasibility. The experimentation involves developing a reasonable hypothesis and designing and performing an experiment to test this hypothesis.

For Dropbox Client Application implementation, we used Python 2.7.5+ and Python SDK for Dropbox API v2. The SDK was downloaded via github.com and installed using the provided setup script.

In order to create an application that is able to upload and download files to a Dropbox folder, one has to register a Dropbox API application. This is accomplshed using App console. Developer has to login before creating an app. First step when creating the application on Dropbox developers page, is to select the API that is going to be used. There are two available options: Dropbox API and Dropbox Business API. Since we want to measure latency using a regular Dropbox account, we chosen Dropbox API. Step 2 is to choose the type of access one needs: access to a single folder specifically for the current app and access to all files and folders in the connected user Dropbox folder. For this application, we used app folder access. Step 3 and the last step is to choose a name for the application. After application creation, we generated an access token that is going yo be used inside the implementation of the client application so we can access API v2. The account token can only be used to access the account of the user registered when creating the application.

After creating the application, a new folder, named Apps, is added in the Dropbox folder of the registered user. Inside it, there is another directory that has the name user introduced when creating the application. This is the folder that is going to be used when interacting with the application we implement.

For simplicity, the project contains only one file, Main.py. We implemented the funtionalities stated in Design chapter pursuing the following workflow:

* Instantiate Dropbox
* Create new folder that will contain the files to be uploaded
* Generate 10 binary files inside the folder with sizes from 14.3 MB to roughly 143.2 MB
* For each file in the created folder, print size and upload it to Dropbox; measure time needed only for upload
* For each file previously uploaded, print size and download it to local directory; measure time needed for download

To make calls to the API, a Dropbox instance is required. When instantiating Dropbox, the access token previously generated has to be passed as an argument:

dbx = dropbox.Dropbox(TOKEN)

To test if the right account was linked and if the instantiation have not failed, we printed the linked Dropbox account email:

print 'Linked Dropbox account: ', dbx.users\_get\_current\_account().email

Dropbox SDK exposes functions for both upload and download. For upload, we used files\_upload function:

**files\_upload(f,path,mode=dropbox.files.WriteMode.overwrite,autorename=True,client\_modified=None,mute=True)**

Data represents the content of the file to be uploaded and is obtained using read() method. file\_path represents the path in the user’s Dropbox to save the file. mode selects what to do if the file already exists. If there’s a conflict, as determined by **mode**, have the Dropbox server try to autorename the file to avoid conflict by setting autorename parameter to true. client\_modified records timestamps of user requests. Mute parameter configures whether or not a notification should be sent to the user when modification are performed.

For download, the exposed function is more straight-forward:

md, res = dbx.files\_download(file\_path)

The file\_path is the path of the document to be downloaded. The metod returns downloaded file metadata, md, and a response, res. We use response to get the size of the downloaded file:

len(res.content)

Time is measured using time.time() method before and after we call files\_upload and files\_download. The actual time for performing an operation is calculated as the difference between start time and end time.

The application was tested on a machine running Ubuntu 13.10 32-bit operating system, 4GB RAM, Intel® Core™ i5 CPU M 480 @ 2.67GHz × 4 Processor and 83.3GB disk space. A internet connection speed test was performed before testing the application, using sppedtest.com, showing a download speed of 26.87Mbps and 25.40Mbps upload speed.

Uploading 100.22 MB in 59.325s

Uploading 71.58 MB in 34.629s

Uploading 28.63 MB in 23.600s

Uploading 114.53 MB in 62.932s

Uploading 85.90 MB in 39.810s

Uploading 128.85 MB in 61.514s

Uploading 143.17 MB in 70.035s

Uploading 57.27 MB in 30.091s

Uploading 14.32 MB in 8.544s

Uploading 42.95 MB in 24.289s

Downloading file\_7 in 1.880s

Downloading file\_5 in 1.227s

Downloading file\_2 in 1.215s

Downloading file\_8 in 1.315s

Downloading file\_6 in 1.251s

Downloading file\_9 in 1.197s

Downloading file\_10 in 1.225s

Downloading file\_4 in 1.179s

Downloading file\_1 in 1.244s

Downloading file\_3 in 1.081s