README

Team Name: GoData (TEAM C)

Team Members:

Anahita Bhiwandiwalla:	ab3744
Digvijay Singh:	ds3161
Kuber Kaul:	kk2872
Payal Rani:	pr2417

Project Proposal: GoDataProjectProposal.pdf

Presentation and Demo Files:

Presentation slides with Web and Mobile Client Demo: http://prezi.com/pvk30uupyabj/copy-of-godata/

Desktop Client Demo Video files:
https://www.dropbox.com/s/89576khwsg78iyl/GoDataDesktopClientDemo1%20.mp4
https://www.dropbox.com/s/0ijdm4l1szg95aa/GoDataDesktopClientDemo2.mp4

GoData Application URL: http://godatacloud.appspot.com

Source Code Files Included:

1. README.pdf: This file

2. GoData web application contains the following source files:

   a. app.yaml : Configuration file for Google App Engine

   b. Apriori1.py: Implementation of the apriori algorithm to predict the files that might be requested by the user. The output of this script is a list of file ids which is used to pre-fetch these files.

   c. Dbtables.py: Creates three tables: FileInfo, Photo, and Apriori to store data in DataStore

   d. Delete.py: Script used to delete a file from the BlobStore

   e. Filedownload.py: This script downloads a file from BlobStore.

   f. Fileupload.py: This script saves the uploaded file to BlobStore.

   g. Thumbnail.py: This script displays thumbnail for mobile clients.

h. Uploadfile.py: Main file for GoData. It takes data from apriori1.py to get the files for pre-fetching. It checks for the type of client and redirects the request to upload.html.

i. images/header.png: Header image for GoData web page

j. templates/upload.html: Homepage of GoData website

k. templates/thumbnail.html: Web page to display thumbnail for mobile client

3. GoData desktop client application contains the following source files:

   client.py : This has the client side code of our Desktop application.It connects to the GoData servers after authenticating with users' Google Accounts.

   client2.py : This contains the GUI version of client -side code using Tkinter (A GUI Library for Python)

   Clientserver.py: This is the corresponding server for the above client,residing on the Google App Engine..

   Cache2.py : This has the implementation of Randomized Marking Algorithm which builds an intelligent Cache Replacement Policy for the local/Desktop Client

Description

1. GoData Web Application:

The workflow can be described as:

The user authenticates through Google authentication APIs

The web application enables a user to upload, view and delete their files.

For mobile clients, the application displays an option to view thumbnail to reduce network bandwidth.

It saves the files on BlobStore and maintains information about the files uploaded by the users on DataStore.

Whenever the user uploads or views a file, the file is stored with the timestamp value to a table in DataStore.

This table is used by the Apriori algorithm to predict the files that may be requested by the user.

The list of file ids returned by apriori algorithm is used to intelligently pre-fetch the files using the pre-fetch feature of HTML5.

When the user accesses the file which has been pre-fetched, it is served from the browser cache.

<u>Prefetching:</u> Apriori Algorithm- We have implemented a prefetching mechanism using the Apriori Algorithm which is popularly used for mining frequent item-sets. We do the following in this algorithm:

We keep a track of when each file has been accessed i.e. uploaded or downloaded/viewed.

1. We have defined each 'session' to be five minute intervals within an hour and we keep a track of files accessed in each session to understand any correlations in file access.
2. We have built on the assumption of apriori algorithm that any subset of a frequent item-set must be frequent. Hence in our context, if fileA and fiileB have been accessed multiple times in the same session they are most likely to be accessed together again. Hence, these files are pre-fetched in the browser's cache.
3. The apriori algorithm is run each time the user logs in to the system or when the application has been refreshed indicating the currently most likely combination of files to be accessed.
4. The size of the cache and duration of each session could be modified depending on the need of the application.
5. We have supported the browser pre-fetching for popularly used browsers- Google Chrome, Mozilla Firefox and Internet Explorer.

2. GoData Desktop Client:

The workflow can be described as:

The user configures the client and connects with a gmail account.

The client enables a user to upload, view and delete their files.

It saves the files on BlobStore and maintains information about the files uploaded by the users on DataStore.

Whenever the user uploads or views a file, the log file is updated.

This log file is used to implement Randomized Marking Algorithm to maintain local cache.

When the user requests a file which exists in the cache, it is served from the local cache.

<u>Cache Management:</u> Randomized Marking Algorithm

1. Initially the cache is empty
2. When a user uploads or downloads a file, if it does not exist in cache then add it to the cache and mark the corresponding bit.

3. When a user uploads or downloads a file, if the file is in cache, mark the corresponding bit as set and serve the file from the cache.
4. If there is a cache miss and the cache is full, then randomly choose one of the files whose bit is unmarked and remove it from the cache.
5. If all the files in the cache have corresponding bit marked, then unmark all bit and start another phase.

Evaluation: On a dummy dataset consisting of 15 files and with a cache of size 5, we ran the algorithm for 15 sample queries which were requested for 10 times with random cache initialization each time we got the average case cache hit score to be about 60% and best case to be about 70%.