

Distributed Cloud Storage Application

Group # 3

Introduction/Overview:

Our Product is a web-based application built on top of *Ruby on Rails* and *MySQL database*. It basically provides 3 major functionalities:

- I. Firstly, Our App manages all of a user's major online social network and Email Accounts: Facebook, Twitter, Stack Overflow, GitHub, LinkedIn, Google Plus+, Gmail. This is shown below in Figure 1:

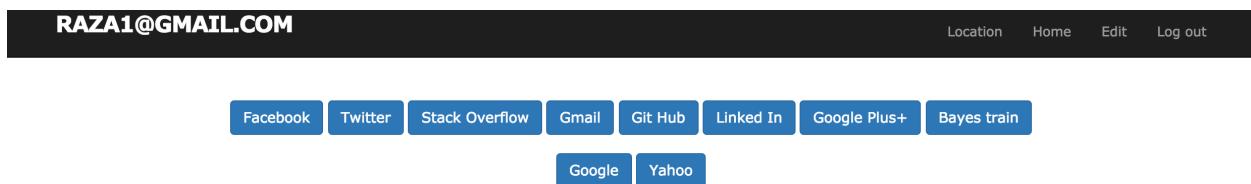


Figure 1: App Interface for a User's Social Networks

- II. Secondly, Our App categorizes each file that a user uploads into one of 4 categories using a famous machine learning text classification algorithm: Naïve Bayes Classifier. While uploading a file, the user can either himself choose the category of the file from a dropdown menu or choose the option 'machine recommendation'. When this option is chosen, Naïve Bayes algorithm is executed, and the category having the highest similarity score with the document is assigned. This is shown below in Figure 2.

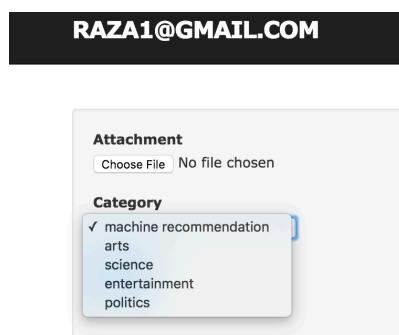


Figure 2: Category options for a file. Choosing 'machine recommendation' runs the Naïve Bayes Algorithm

- III. Thirdly, Our App provides 4 major ways a file can be downloaded. Each of them is a distributed Join Algorithm:

- ✓ Semi Join

- ✓ Bloom Join
- ✓ Parallel Semi Join
- ✓ Parallel Bloom Join

The screenshot shows a user interface for managing files across distributed databases. At the top, there's a header bar with the email 'RAZA1@GMAIL.COM' and navigation links for Location, Home, Edit, and Log out. Below the header is a row of social media and service links: Facebook, Twitter, Stack Overflow, Gmail, Git Hub, Linked In, Google Plus+, Bayes train, Google, and Yahoo. The main content area is titled 'Your Files on Distributed Databases:' and contains a table listing four files (cv.txt, file.txt, art1.txt, fun10.txt) with their respective categories (politics, arts, politics, politics). For each file, there are four blue buttons labeled 'Download Using Semi-Join', 'Download Using Bloom-Join', 'Parallel Semi-Join', and 'Parallel Bloom-Join', followed by a red 'Delete' button.

Figure 3: Each File can be downloaded using 4 major distributed join algorithms

Session Cookies:

When a user logs into his/her account, a session cookie is created that is maintained until the user explicitly logs out. A particular user cannot visit other user's profile.

The screenshot displays two side-by-side forms within a 'CLIENT APP' interface. Both forms have a dark header bar with 'Home', 'Help', and 'Sign up' links. The left form is titled 'Log in' and contains fields for 'Email' and 'Password', with a 'Log in' button. Below the form is a link 'New user? Sign up now!'. The right form is titled 'Sign Up' and also contains fields for 'Email' and 'Password', with a 'Sign up!' button. Below this form is a link 'Have an Account? Log In now!'

Semi Join Algorithm:

- Objective is to reduce the number of tuples in a relation before transferring it to another site.
- When a request Download Using Semi-Join comes in,
 - ❖ The Client App sends the user's email and the filename to all the 4 Server Node,
 - ❖ Fetches the response data from each of the Servers,
 - ❖ Combines all the data fetched, and finally,
 - ❖ Creates a new file and saves it on the user's machine.

- All the communication between the clients and servers is through a TCP socket.

Bloom Join Algorithm:

- When a request [Download Using Bloom-Join](#) comes in,
 - ❖ The Client App calculates the hash of each user email + filename,
 - ❖ Mods that value with k (in our case k=100),
 - ❖ Sets the value to 1 in bit-vector where index = hash value. (Please Note that the bit-vector is of size 100).
 - ❖ Sends the bit-vector to all the 4 Server Node.
 - ❖ Each Server Node similarly calculates the hash value for each (email, filename) pair it has stored in its database, and if a particular index(= hash value) in the bit-vector has value 1, then it sends the data for that element to the Client App.
 - ❖ The Client App fetches the response data from each of the Servers,
 - ❖ Combines all the data fetched and finally,
 - ❖ Creates a new file and saves it on the user's machine.
- The hash algorithm and calculation of bit vector is shown below:

```

703     for i in 0..(emails.length - 1)
704         email = emails[i]
705         filename = filenames[i]
706         str = email + filename
707         hash_value = 0
708         for j in 0..(str.length - 1)
709             hash_value = hash_value + (j*str[j].ord)
710         end
711         hash_value = hash_value % k
712         bit_vector[hash_value] = '1'
713     end
714     str_bit_vector = ""
715     for b in bit_vector
716         str_bit_vector = str_bit_vector + b +
717     end
  
```

- The hash vector is converted to a string before it is sent to the server nodes.

Parallel Semi Join & Parallel Bloom Join Algorithms:

- When the request [Parallel Semi-Join](#) and [Parallel Bloom-Join](#) comes along, the Client App executes the Parallel Semi Join and Parallel Bloom Join Algorithm to download the file.
- The only difference with regular Semi Join and regular Bloom Join is that the Parallel Algorithm spans 4 threads and sends the request to each server simultaneously and fetching the data from each server in a parallel fashion. This results in a huge increase in performance as shown in Figure 5.

Work flow of File Uploading/Downloading:



When a File is uploaded, the Client App divides the File into 4 pieces and sends each piece to each of the Server Nodes. Server Node is a ruby program which has a TCP server socket listening on a specific port. Each Server Node is running on a separate computer. For the communication between client and server nodes to be possible:

- EITHER, the server nodes are connected to the public LANs such that each server computer has its public IP equal to the private IP. In this case, server nodes and client can be as far from each other as you will.
- OR, the client and all the server nodes are behind the same Wifi network. (We showed this in the Project Demo in class).

Logged-In User's Location Information:

We also provide user with his/her IP address from which they log-in from, and shows their location on a google map, as shown below in Figure 4.

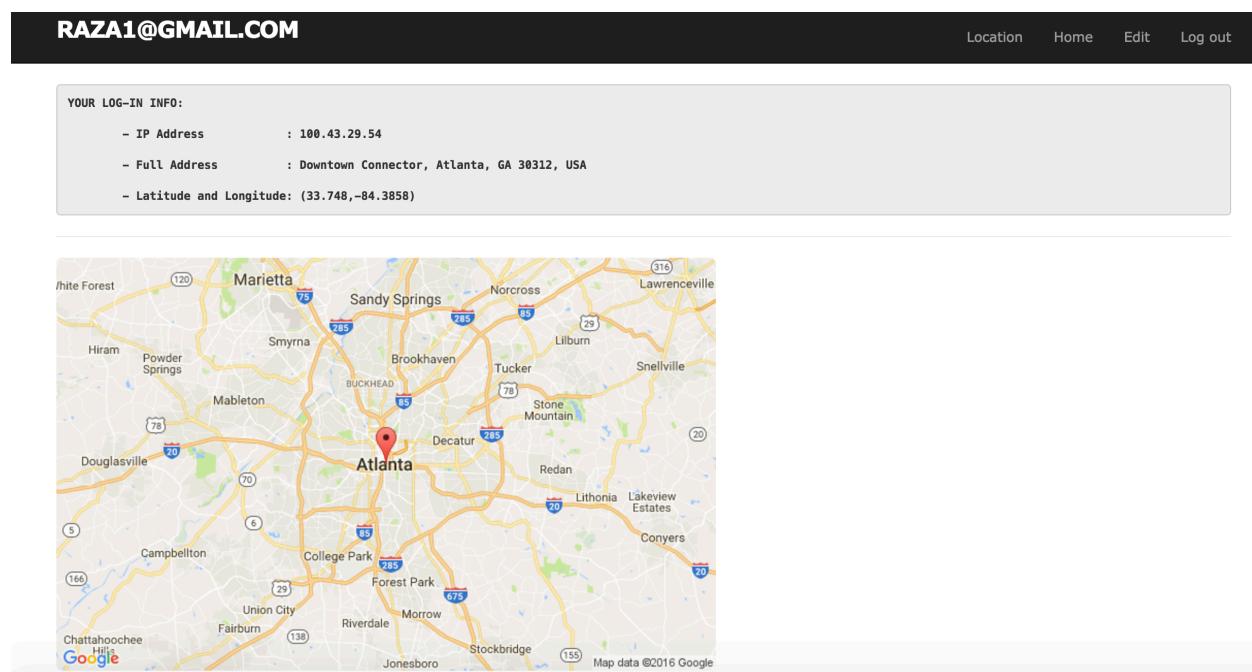


Figure 4: IP address, location address, and location on Map of the Logged-In User

All Social Networks Under One Roof:

The image displays four separate browser windows, each showing a different social media or networking platform. The platforms are:

- RAZA1@GMAIL.COM**: A dashboard showing a profile picture, bio, and statistics (Tweets: 46, Following: 44, Followers: 6). It also displays a news feed from BBC News (World) and an advertisement for Amazon's 12 Days of Deals.
- RAZA1@GMAIL.COM**: A post from "The Wall Street Journal" with 151 likes, 35 comments, and 2 shares. The post includes a banner for "EXCLUSIVE STUDENT OFFER NEW DIGITAL PACK AVAILABLE ONLY \$1 A WEEK".
- RAZA1@GMAIL.COM**: The Stack Overflow homepage, featuring a developer story and a list of top questions. Examples include "How to get id from shared object?", "500 Server Error from SignalR Hub", and "Measure elapsed time in swift".
- RAZA1@GMAIL.COM**: A GitHub profile for "raza115" showing an overview, repositories, stars, followers, and following. Popular repositories listed include "ExamSharingNodeJS", "NodeJS-TicTacToe-multigame", "SPROJ-Collision-Networks", and "Reverse-Engineering-Android-Applications".

Testing:

We wrote a python script using selenium library. The script would go to our website, log in with an email and password, and subsequently download each of the 19 files through each of the 4 methods: Semi-Join, Bloom-Join, Parallel Semi-Join, and, Parallel Bloom-Join. It would then note down the Communication Time Taken for each method for each file. The file sizes range from 59 Bytes to 10 Mega Bytes.

Graph is shown below:

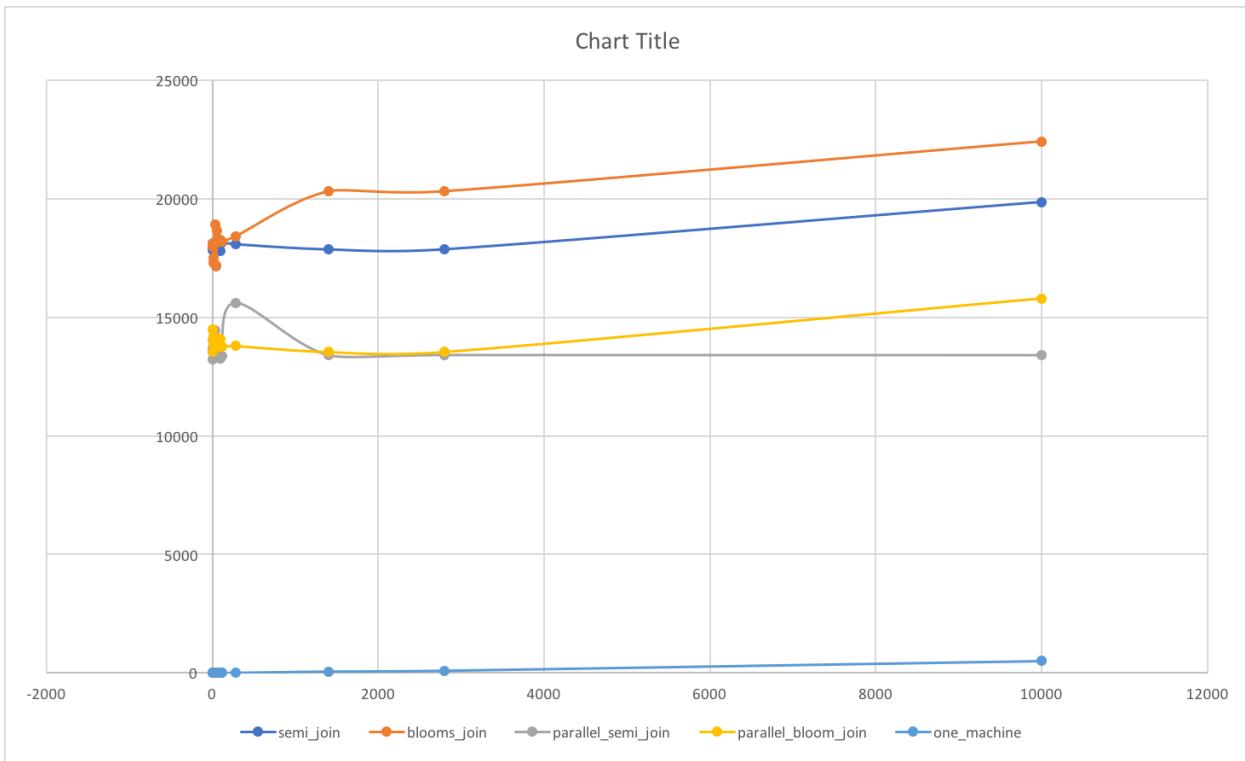


Figure 5: Performance Comparison of the 4 Join Algorithms