

# An Open Source Direct Messaging and Enhanced Recommendation System for Yioop

A Project Report

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

Of the Requirements for the

Degree of Master of Science

By

Aniruddha Dinesh Mallya

Dec, 2021

©2021

Aniruddha Dinesh Mallya

ALL RIGHTS RESERVED

# An Open Source Direct Messaging and Enhanced Recommendation System for Yioop

By

Aniruddha Dinesh Mallya

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSE STATE UNIVERSITY

MAY 2021

Dr. Christopher Pollett

Department of Computer Science

Dr. Katerina Potika

Department of Computer Science

Dr. William Andreopoulos

Department of Computer Science

## **ACKNOWLEDGMENT**

First, I want to thank my project advisor, Dr. Chris Pollett, for all his guidance, patience and motivation throughout the past year. I want to express my sincere appreciation for the opportunity to work with someone as humble and knowledgeable as him.

My sincere gratitude also goes to, Dr. Katerina Potika and Dr. William Andreopoulos, esteemed members of my defense committee. I am also grateful to all the CS faculty for showing their patience, understanding and support towards me during my graduate study.

Finally, I would like to thank my mother, Asha Mallya, for her constant support and faith in me even during the tough times, my friends, Priyam Dhanuka, for his valuable knowledge of the tech industry and using it to guide me in my work, Smridhi Seth, for her loving care and support throughout my studies and to all others who helped me throughout the semester, my heartfelt thanks.

## ABSTRACT

Recommendation systems and direct messaging systems are two popular components of web portals. A recommendation system is an information filtering system that seeks to predict the "rating" or "preference" a user would give to an item and a direct messaging system allows private communication between users of any platform. Yioop, is an open source, PHP search engine and web portal that can be configured to allow users to create discussion groups, blogs, wikis etc.

In this project, we expanded on Yioop's group system so that every user now has a personal group. Personal groups were then used to add user clipboards to the wiki systems and were used to create a viable direct messaging system in Yioop. Next, we have improved upon the current recommendation system for Yioop where given a user's history, threads and groups are suggested to a user. Yioop uses the concept of term frequency and inverse document frequency to provide recommendations, so we added upon this by creating a new recommendation system that uses Hash2Vec. In our experiments we conducted some load tests on our DM system's database and using the chi-squared test we hypothesize a linear execution time in terms of database latency vs volume of data sent by multiple users to our system and we also compared the accuracy between the old and new recommendation systems and saw an improvement in the avg. F1 measure by 60.28%.

**Index Terms: Yioop, Web-pages, Direct-Messaging, Recommendation Systems.**

## TABLE OF CONTENTS

1. INTRODUCTION .....	5
2. BACKGROUND .....	7
2.1 Yioop Search Engine .....	7
2.2 Related Work .....	8
3. PRELIMINARY WORK .....	9
4. DESIGN FOR DIRECT MESSAGING .....	12
5. IMPLEMENTATION OF DIRECT MESSAGING .....	14
5.1 Experiments .....	19
6. YIOOP'S RECOMMENDATION SYSTEM.....	22
6.1 TD-IDF Brief Synopsis.....	22
6.1.1 Term Frequency .....	22
6.1.2 Inverse Document Frequency .....	24
6.1.3 TF * IDF to Calculate Weights .....	26
6.2 Recommending Threads and Groups in Yioop.....	27
6.2.1 Computing TF for Threads .....	28
6.2.2 Computing TF for Users .....	28
6.2.3 Computing IDF for Threads.....	29
6.2.4 Computing IDF for Users .....	29
6.2.5 TF-IDF weights for Threads .....	30
6.2.6 Threads and User Cosine Similarity .....	30
6.3 Group Recommendations.....	31
7. ENHANCING YIOOP'S RECOMMENDATION SYSTEM .....	32
7.1 Word Embeddings .....	32
7.2 HASH2VEC.....	32
7.3 Experiments .....	37
8. CONCLUSION .....	39
BIBLIOGRAPHY .....	40

## LIST OF FIGURES

Fig. 1 USERS and USER_GROUP Table Schemas .....	10
Fig. 2 GROUP_ITEM and SOCIAL_GROUPS Table Schemas .....	10
Fig. 3 Personal Chat Group Name for Users .....	12
Fig. 4 Connect Dropdown for Users .....	13
Fig. 5 Public Posts and Chats Dropdown for Users .....	13
Fig. 6 Class Diagram for Direct Messaging .....	14
Fig. 7 Flow Chart for Direct Messaging .....	15
Fig. 8 Use case when user has no friends .....	16
Fig. 9 USER_GROUP Table to establish connection .....	16
Fig. 10 Use case when both users are not connections of each other .....	17
Fig. 11 GROUP_ITEM Table View for Direct Messaging .....	17
Fig. 12 Use case when both users are connected .....	18
Fig. 12. (a) Multiple Users Execution Time with Standard Deviation .....	19
Fig. 12 (b). Database Latency vs Volume of Data Sent by Multiple Users .....	21
Fig. 13 ITEM_TERM_FREQUENCY .....	28
Fig. 14 USER_TERM_FREQUENCY .....	29
Fig. 15 ITEM_TERM_WEIGHTS Table .....	30
Fig. 16 USER_TERM_WEIGHTS Table .....	30
Fig. 17 ITEM_RECOMMENDATION Table .....	31
Fig. 18 Hashing .....	33
Fig. 19 HASH2VEC Table .....	35
Fig. 20 Improved Scores for Recommendation Table .....	35
Fig. 21 Result from Old Recommendation Table .....	36
Fig. 22 Result from New Recommendation Table .....	36

## LIST OF TABLES

Table 1 Document 1 Term Frequency .....	23
Table 2 Document 2 Term Frequency .....	23
Table 3 Document 3 Term Frequency .....	23
Table 4 Document 1 Log Term Frequency .....	24
Table 5 Document 2 Log Term Frequency .....	24
Table 6 Document 3 Log Term Frequency .....	24
Table 7 Inverse Document Frequency of all words in given corpus .....	25
Table 8 Documents Weighted by Query Terms .....	27
Table 9 Precision and Recall Results .....	38



## 1. INTRODUCTION

Yioop is an open source implementation that acts as a search engine and web portal. As a web portal it lacks some features like Direct messaging (DM). DM is a type of technology that allows one to chat online in real time over any type of computer network like the Internet. Two or more individuals send messages over a network when they each input text and trigger a transmission to the recipient(s). The primary difference between direct messaging and email is that conversations occur in real time, i.e., instantly. DM is a feature shared by all social media platforms. Besides being used for personal purposes, DM can help achieve business goals for example a company can easily interact with clients with DM by sending custom messages about other business collaborations, advertisements or for simply requesting feedback. and for these reasons this feature is being added to a Yioop.

It is important to first note that many social networking websites like Facebook, Twitter, Instagram provide the capability of creating groups dynamically. Looking at Facebook for example a group creation option is available on the homepage of the web application. These groups can be dynamically created by a user, allowing them to connect with different people, such as family, friends, classmates or people who share the same interests. Yioop precisely provides such a feature. Depending on the role of the user within the group (owner vs. user), members of a group can be given access control and we further extend this access control to allow users to directly send messages to other users with whom they are connected.

In a recommendation system, users are given suggestions as to which news articles to browse, which movies to watch, which restaurants to eat at, what things to do in a group etc. in this manner we can potentially find the information most relevant to us, rather than letting us skim through or search through a series of items. Recommendation systems in companies like Google, Youtube or Facebook use TF-IDF as a base [12] [13] to provide suggestions for news articles or different medias like music/videos. In this project, we look at a similarly built recommendation system for Yioop using its open source discussion groups. Prior to improving this recommendation system, we examined the internal working of Yioop's recommendation engine. Initially, the system was built using collaborative filtering but it ran into the risk

of only recommending popular threads [14]. Next, the recommendation model was extended to make predictions using term frequency (TF) and the inverse document frequency (IDF) of users and threads. Here, we observed that the TF-IDF of words with respect to user and threads were only considered while recommending a thread but contextualizing the word to potentially improve thread recommendations was missing. So we decided to implement Hash2Vec in order to be able to contextualize the words. We considered traditional neural networks like word2Vec or Glove to contextualize words however the problem is that it is expensive to train these models and in order to be able to use these models in quick an efficient fashion big companies end up relying on cloud based services and permanently rely on such external services. The novelty of Hash2Vec lies in the fact that it uses a non-neural network model and so it does not require any training and yet it shows promising results — detailed further in the Chapter 7 — making Yioop ideal to use for small scale organizations that cannot afford expensive cloud services but still maintaining its large scale functionality.

This report is organized as follows: Chapter 2 gives some background about Yioop and we look at some important background related to the direct messaging and recommender systems. Chapter 3 talks about the preliminary work that went into understanding and setting up Yioop. Chapter 4 details what the different tables in Yioop are and how we take advantage of it to setup our direct messaging system. Chapter 5 goes over some of the basics in the Yioop recommender system and how it is built in Yioop. Chapter 6 discusses about how the current recommendation system can be improved with a technique called hashing and Chapter 7 discusses some experiments performed to test the systems effectiveness. Chapter 8 gives a conclusion to the project and discusses possible future work for it.

## 2. BACKGROUND

### 2.1 Yioop Search Engine

In this chapter we will look at the Yioop's open discussion group functionality and try to understand a little bit of how this functionality works. Firstly, Yioop is written in PHP and released under GPLv3 license [2]. It is available as a download for free. In addition to a regular search engine, the Yioop search engine also allows users to search for URLs that have been pre-defined by them, i.e., when a crawl is performed in Yioop and the user can select the URLs or URL ranges they want to be searched for the search results. Yioop uses its own model-view-controller framework and has an easy-to-use interface. All documentation and resources on Yioop are available on the Seekquarry website [1].

Now, coming to Yioop's group functionality feature, in addition to creating groups users can create pages within a group with different access controls on these features. Different groups or different users within a group can share pages amongst each other. Here, each group or user can create its own page for the articles they have written and these pages can be used just like wikis.

Despite all these features Yioop leaves a lot to be desired for example if users of the platform would like to communicate directly with other users rather than openly ask something on a discussion forum like on threads of a public group where others can see this conversation. This is where a direct messaging system would be highly desirable allowing users to privately communicate with each other and is introduced as part of this project.

Additionally, we know users of Yioop can create discussions groups and start new threads within those groups to share information with other users. Using this information and based on the threads or groups a user has viewed previously, Yioop's recommendation engine suggests threads or groups they might be interested in. So, users are recommended two items through the recommendation system, i.e., Groups and Threads.

## **2.2 Related Work**

In their study, Argerich et. al [8] proposed a new way to derive the embedding vectors for words that did not involve any neural networks; they utilized a hashing function. We will look at this hashing concept called Hash2Vec to possibly improve the thread and group suggestions in Yioop. Now say, we use a Continuous Bag of Words (CBOW) model for a million words it makes a co-occurrence matrix of size million by million giving it a space complexity of  $O(n^2)$  and it also have an expensive training time to process all million words in their vectorized forms. According to Argerich et. al in Glove, word2vec, the vectors were generated with a predefined length to keep this matrix optimized, since a massive matrix is impractical to process. While the hashing technique does not require training, it will require some processing time however it's space complexity is  $O(nk)$ ,  $n$  = numbers of words in corpus and  $k$  = a fixed dimensionality which can be small, showing an immediate improvement in comparison to the word2vec/Glove training. The Hash2Vec resultant word embeddings of similar words are on par with Glove's word embeddings [8], thus making it favorable to implement in PHP and prevent poor performance that PHP can face with large scale data [9].

### 3. PRELIMINARY WORK

In this chapter we will look at what and how a direct messaging system works and some of the initial work that went into integrating it into Yioop. A DM system provides real time transmission of information between users of the system privately through a medium like the internet. Messages can even be sent to users not logged in (offline messages), eliminating some of the differences between direct messaging and email [6]. Similarly, Yioop chat system aims to make text conversations available to be saved and used later.

Before tackling the problem of direct messaging we looked at some instant messaging protocols used by companies like WhatsApp and iMessenger both of which use the XMPP protocol [3]. However, since this leads to dependencies on external libraries we decided to rely on a simple AJAX and database interaction to simulate the direct messaging. We will first look at how Yioop manages its groups feature. Yioop follows the model-view-controller (MVC) software design pattern, and the Manage Group functionality allows users to add, delete or view groups. We are interested in the add and view groups methods, essentially it works by forwarding requests to the controller, the controller assigns the request to a relevant action and then passes it to the model. Models perform the necessary actions and send database responses to the controller, which sends the response and renders the applicable view on the browser.

The database tables are designed such that they store and retrieve the data efficiently. We are interested in the following tables: USERS, USER\_GROUP, GROUP\_ITEM and SOCIAL\_GROUPS. The columns in the four tables mentioned are shown in Fig. 1 and Fig. 2.

USERS		USER_GROUP	
<b>user_id</b>	bigint	<b>user_id</b>	bigint
first_name	varchar	<b>group_id</b>	bigint
last_name	varchar	status	bigint
username	varchar	join_date	numeric
email	varchar		
password	varchar		
status	bigint		
creation_time	varchar		
ups	bigint		
downs	bigint		

Fig. 1 USERS and USER\_GROUP Table Schemas

GROUP_ITEM		SOCIAL_GROUPS	
<b>id</b>	int	<b>group_id</b>	int
parent_id	bigint	group_name	varchar
group_id	bigint	created_time	varchar
user_id	bigint	owner_id	bigint
url	varchar	register_type	bigint
title	varchar	member_access	bigint
description	varchar	vote_access	bigint
pubdate	numeric	post_lifetime	bigint
edit_date	numeric		
ups	bigint		
downs	bigint		
type	bigint		

Fig. 2 GROUP\_ITEM and SOCIAL\_GROUPS Table Schemas

The USERS table is updated when a new users registers or is added to the Yioop platform and is given a “user\_id”. The USER\_GROUP table is consists of the given “user\_id” for a user and the “group\_id” of the groups a user is part off. The SOCIAL\_GROUPS table consists of “group\_name” and “user\_id”. Finally GROUP\_ITEM table consists of the “thread\_id” and “group\_id” to identify which threads belong to which group along with the user comments that are part of the thread.

From this we understand that groups access is controlled by assigning “group\_id” to different users in the USER\_GROUP table and comments to a thread are saved in the GOURP\_ITEM table. We take advantage of this in place schema to establish a direct messaging system for Yioop explained in the following chapters.

We even researched Netflix’s recommender system [4] to understand how collaborative filtering works and its different approaches. While Yioop initially was setup with a custom collaborative filtering technique which involved looking at a user’s posts and the user’s views on a thread, it led to recommending trending threads to a user. This was further extended using the td-idf technique with user and thread in mind and which led to more custom recommendations. However, going ahead we will dive deep into the working of this system and point out what can potentially be improved using a simplified word embedding technique called Hash2Vec.

#### 4. DESIGN FOR DIRECT MESSAGING

In this chapter we look at the design idea for a DM system in Yioop. So, earlier we learnt about the different tables present in the database of Yioop and in order to manipulate the tables related to groups, the “GroupModel” class was used and to add users the “UserModel” class was used. Hence for our DM system these models were used to create a pre-defined group called "Personal" through the "AccountaccessComponent" controller class. So when a new user is introduced into the Yioop environment and that user logs in for the first time, the aforementioned pre-defined group is created automatically.

The SOCIAL\_GROUPS table manages the group information for a particular user and one of the constraints for a user to create a group is that the group name needs to be unique. Now since “Personal” is a way of identifying the chat group for a user we needed to add something more to this title to make it more distinguished for a user. We did this by combining the word “Personal” and a user’s unique identifier in Yioop , i.e., “user\_id” in the USER table for example, “Personal\$1”, here \$ is a delimiter and 1 is the user\_id for the admin of this system.

GROUP_ID	GROUP_NAME
328	CS 267 Spring 2021
331	CS 256 Fall 2021
332	CS 152 Fall 2021
333	Personal\$4
334	Personal\$1

Fig. 3 Personal Chat Group Name for Users

Before being able to deal with direct messaging between users first we had to decide on how to allow users to connect with each other. So we decided to allow users to connect with other users through a drop down option as shown in the Fig. 4.



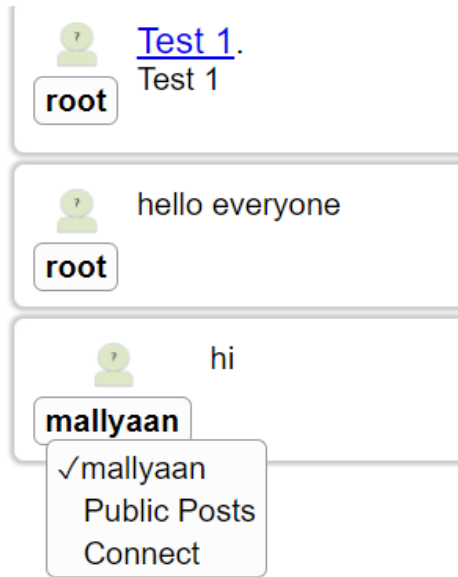


Fig. 4 Connect Dropdown for Users

So different users can connect with each other as long as they are part of a common group.



Fig. 5 Public Posts and Chats Dropdown for Users

Since the “Personal” group is a custom group specially used for DM between people, it appears as the “Chat” option shown above. This dropdown is available to view from any group feed view in Yioop and a user can use this option to access their private chats with other users. “Public Posts” as the name implies shows the posts made by a user and are available for all to see.

## 5. IMPLEMENTATION OF DIRECT MESSAGING

Now we have a solid foundation to move forward with implementing the DM system in Yioop. There are three uses cases for this problem statement, i.e., handling the logic for when a user has no friends, when one user sends a friend request while the other user has not accepted the connection request and finally when both users have accepted the connection requests from each other. Since Yioop follows the MVC model all the logic has to be handled by the controller.

Fig. 6 and Fig. 7 is the design idea and logic flow behind the direct messaging system to be deployed on Yioop. The class diagram in Fig. 6 includes the applied methods but is not limited to it.

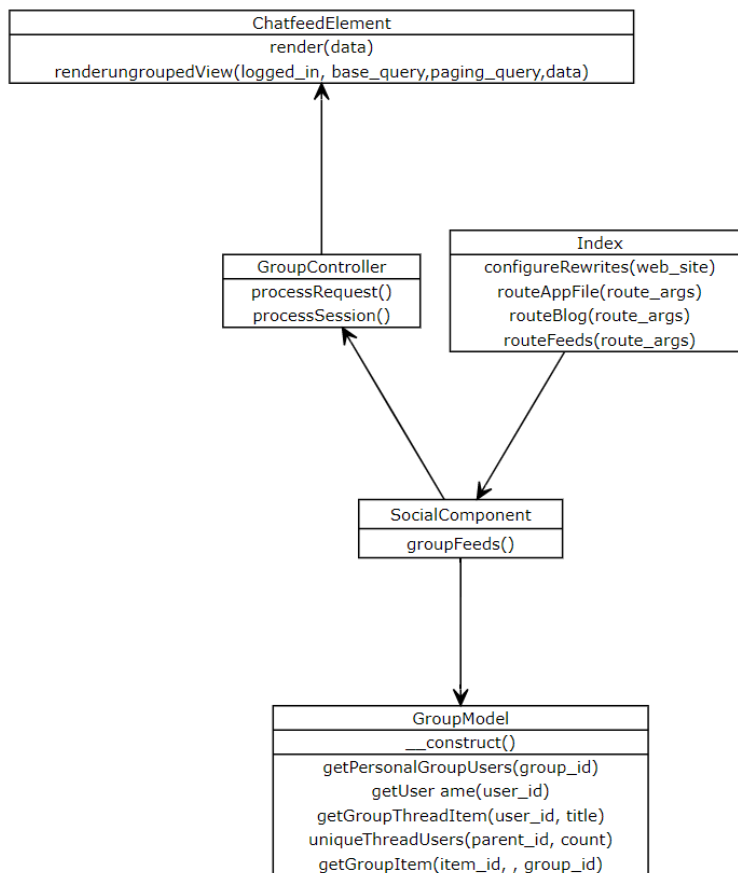


Fig. 6 Class Diagram for Direct Messaging

This DM feature is primarily handled by the groupFeeds() function under the SocialComponent class which processes requests from user under the controller domain. The SocialComponent sends these requests to the GroupModel, which executes the tasks like adding users to groups, deleting groups, etc., and sends back a response. Next it calls the GroupController class that extends the Controller class which further calls the ChatFeedElement class that ultimately renders the view.

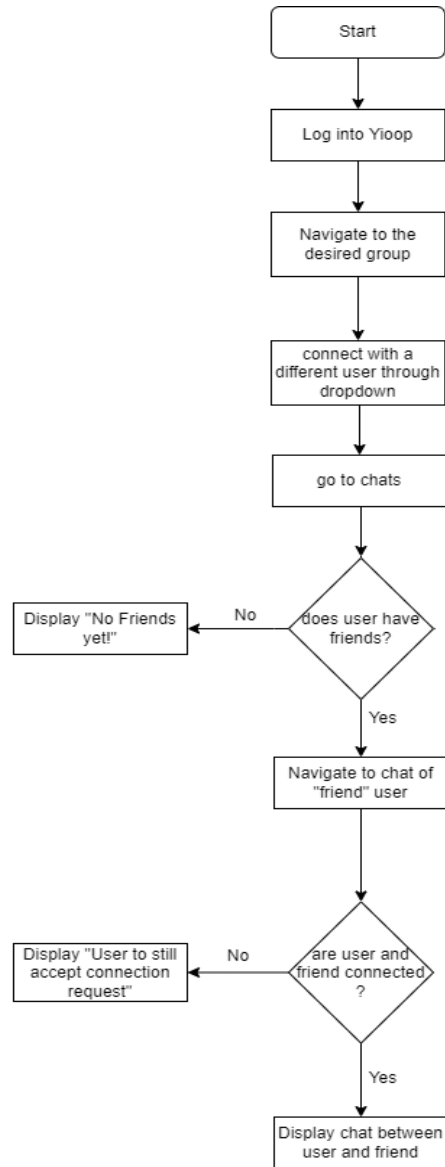


Fig. 7 Flow Chart for Direct Messaging

Looking at use case one, when a user has no connections, i.e., no friends. We simply do this by

checking if a user’s has any friends—equated to threads—as part of their “Personal” group. Fig. 8 shows the result of such a scenario.

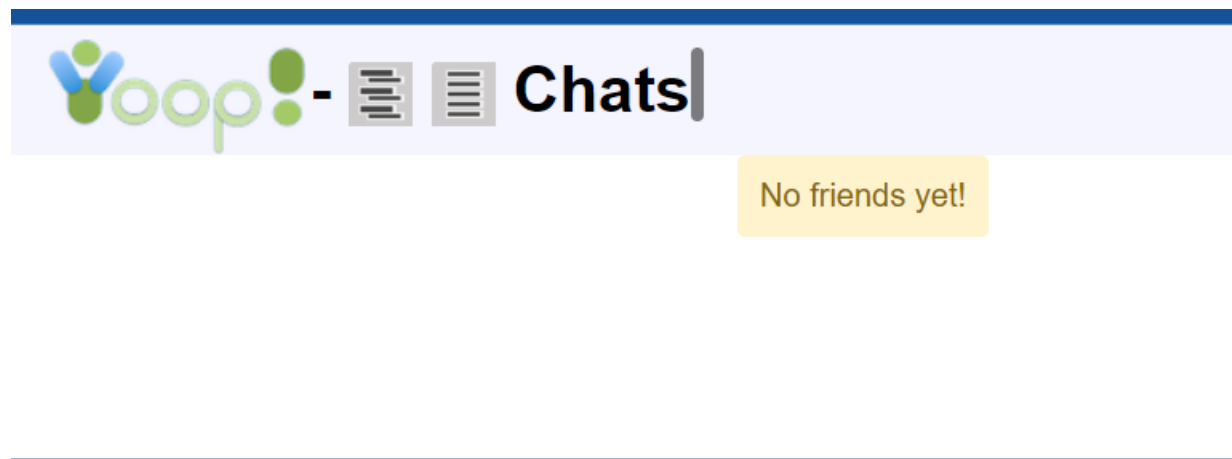


Fig. 8 Use case when user has no friends

Now coming to use case two, when a user sends the connection request to a different user and the connection has not connected with the user then the connection is handled by prompting the user to wait for the connection to connect with the user as well. To do so, first the user gives the connection access to their “Personal” group, this done in the backend database in the USER\_GROUP table. For example looking at the table in Fig. 9,


USER_ID	 GROUP_ID	STATUS	JOIN_DATE
1245	334	1	1638053981
1	334	1	1630269053

Fig. 9 USER\_GROUP Table to establish connection

Here, the “Personal” group for user with “user\_id: 1” is also given access to user with “user\_id: 1245”. Next, we do vice versa, i.e., we check if “Personal” group of “user\_id: 1245” is given access to “user\_id:

1”, if not the Fig. 10 below is displayed.

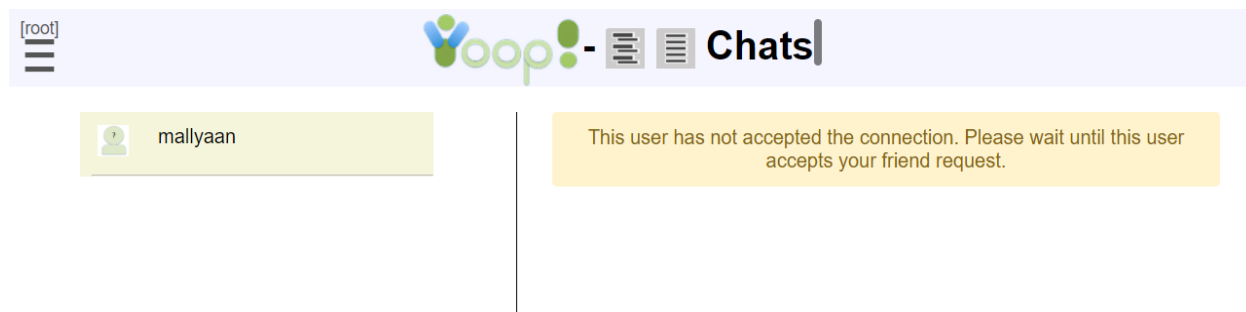


Fig. 10 Use Case when both users are not connections of each other

Final use case three, when both users are connected to each other which is indicated by the USER\_GROUP table as mentioned above, we then have to store the chat between any two users, to do so we use the GROUP\_ITEM table. But since we are dealing with two “Personal” groups of the two users “texting” each other we had to save the “text” for both the groups. The table is as shown in Fig. 11.

ID	PARENT_ID	GROUP_ID	USER_ID	URL	TITLE	DESCRIPTION	PUBDATE	EDIT_DAT	UPS	DOWNS	TYPE
445679	445671	334	1		-- mallyaan	the weather is ...	1638054098	1638054098	0	0	0
445680	445671	357	1245		mallyaan	the weather is ...	1638054098	1638054098	0	0	0
445677	445671	334	1		-- mallyaan	im fine how are ...	1638054070	1638054070	0	0	0
445678	445671	357	1245		mallyaan	im fine how are ...	1638054070	1638054070	0	0	0
445675	445671	357	1245		-- mallyaan	how are you?	1638054030	1638054030	0	0	0
445676	445671	334	1		root	how are you?	1638054030	1638054030	0	0	0
445673	445671	357	1245		-- mallyaan	hi	1638054022	1638054022	0	0	0
445674	445671	334	1		root	hi	1638054022	1638054022	0	0	0
445672	445671	357	1245		root		1638054010	1638054010	0	0	0
445671	445671	334	1		mallyaan		1638053981	1638053981	0	0	0

Fig. 11 GROUP\_ITEM Table View for Direct Messaging

The table in Fig. 11 is displayed as follows to a user.

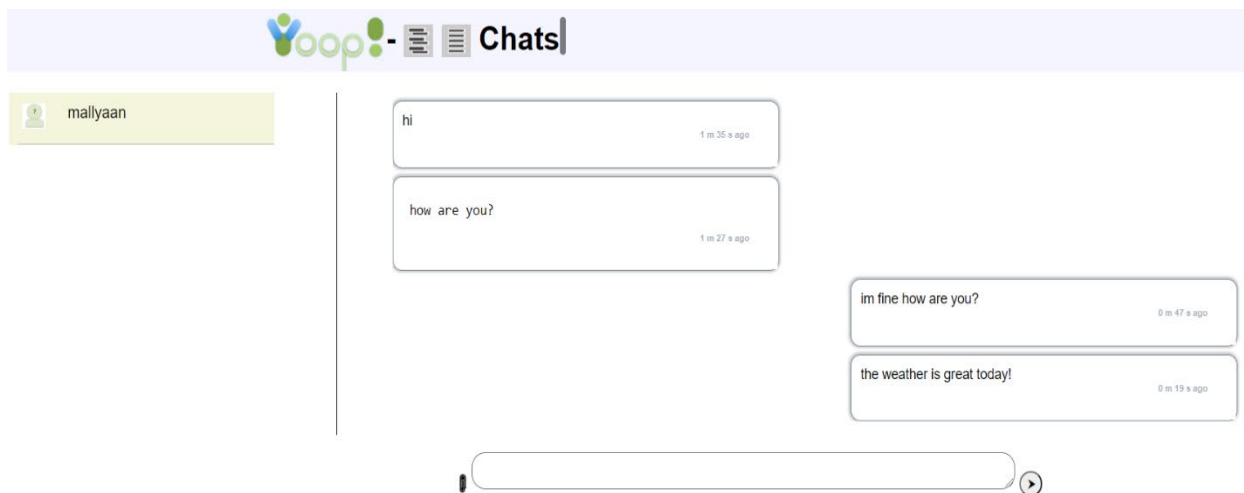


Fig. 12 Use Case when both users are connected

We faced some challenges while creating this DM system. The first experiment we tried was creating a "Personal" group in one click for all existing users. However, due to the number of dependencies between different tables in Yioop's database, this proved to be challenging to handle. We then had to modify the method for each new user. We developed a method for creating groups such that when a user first logged into Yioop we check if the active status for that user was 1. If so, we created the "Personal" group for that user and changed their active status from 1 to 5 in the database. When new features are added to Yioop, this logic will be used to identify it's different versions. A second challenge was to manage the title view of the "Personal" group which displayed a user's full "username" and "user\_id" on different webpages that were dependent on the SOCIAL\_GROUPS table, such as all the groups they are a part of or the menu bar. The first step was to work out that all of the display functionality was controlled by "Element" class under the "views" section on the backend of Yioop and then create a method which could be used across other elements that extend this class to address this issue.

The third challenge was to get two different pages to display on the same page. The DM system uses the "GroupfeedElement" class as a foundation to display results to the user, the logic for a how the page will look when a list of threads in a group is requested is different from when comments by users under a particular thread will look is different and is displayed on two different pages. We had essentially

combine the design for the two separate pages into one page and move things around to get the chat view displayed in Fig. 12.

### 5.1 Experiments

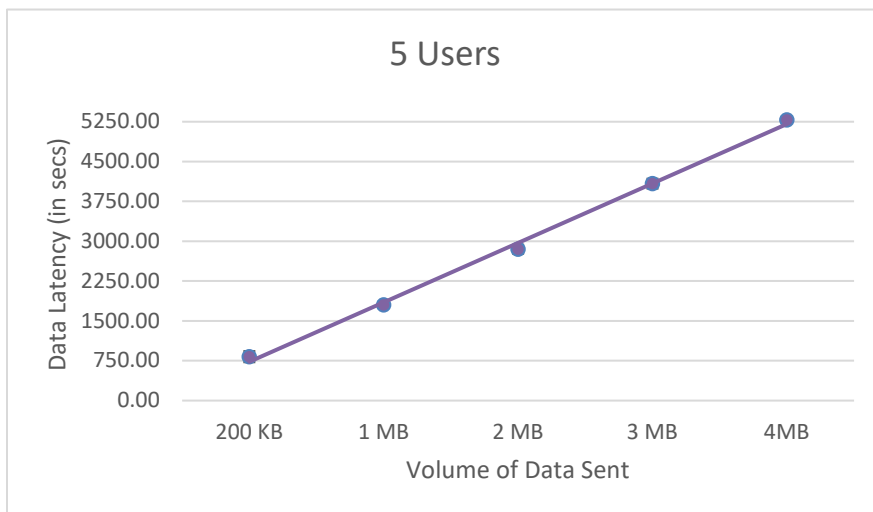
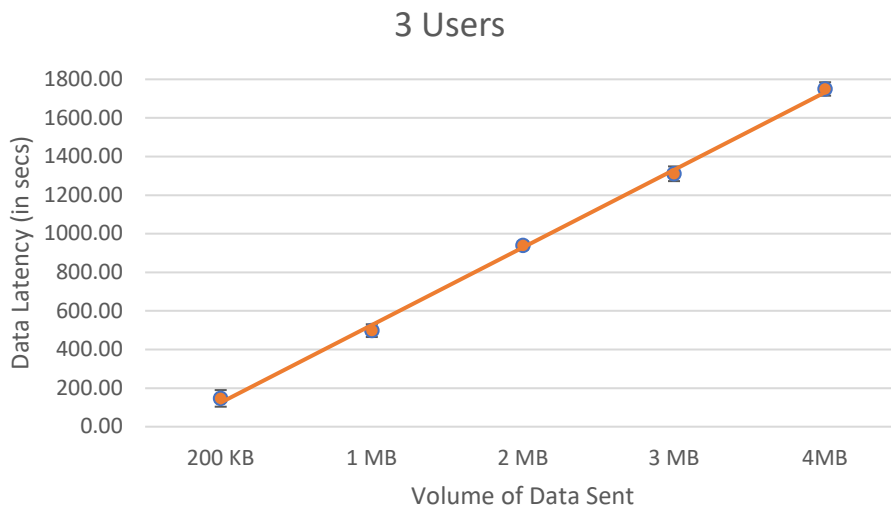
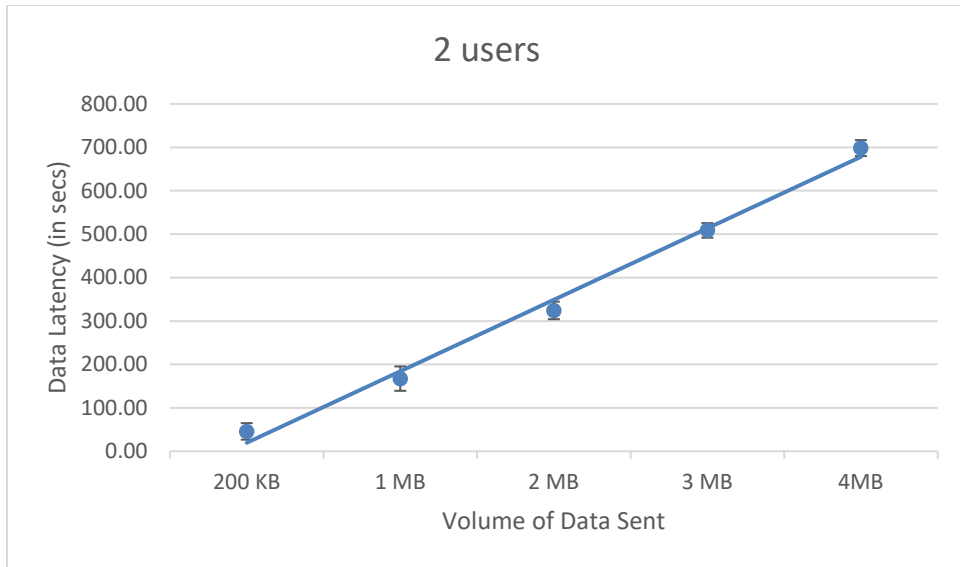
To get an idea of the performance of this implementation we did some load tests on the Yioop backend database. To simulate multiple users we created a program that created instances of multiple insertions on the same local machine. These insertions are meant to also simulate the transactions that take place when users send messages to each other. We timed the programs and the latency information was captured in terms of seconds the table is shown below in Fig. 12. (a).

	2 users	+/-	3 users	+/-	5 users	+/-	10 users	+/-	15 users	+/-
200 KB	45.70	19	146.64	43	826.88	30	1390.90	35	3071.35	37
1 MB	167.27	28	497.64	33	1801.69	58	5790.10	23	12079.14	34
2 MB	324.32	20	938.82	21	2850.18	86	11210.13	35	20010.12	34
3 MB	508.60	17	1310.66	38	4084.04	94	16823.21	29	31095.62	34
4MB	698.23	18	1750.06	35	5280.26	54	22619.18	41	40161.49	27

Fig. 12. (a) Multiple Users Execution Time with Standard Deviation

We calculated the above table by taking the avg. for each set of multiple users.

For example consider 5 users interacting with the database; by running it 5 times we took out the avg. = 826.88 secs and standard deviation = 15. This means in the case of 5 multiple users the total execution time would be anywhere between 796.88 – 856.88 secs around 95% of the time (assuming all systems run fall under the normal distribution curve), i.e., within two standard deviations.





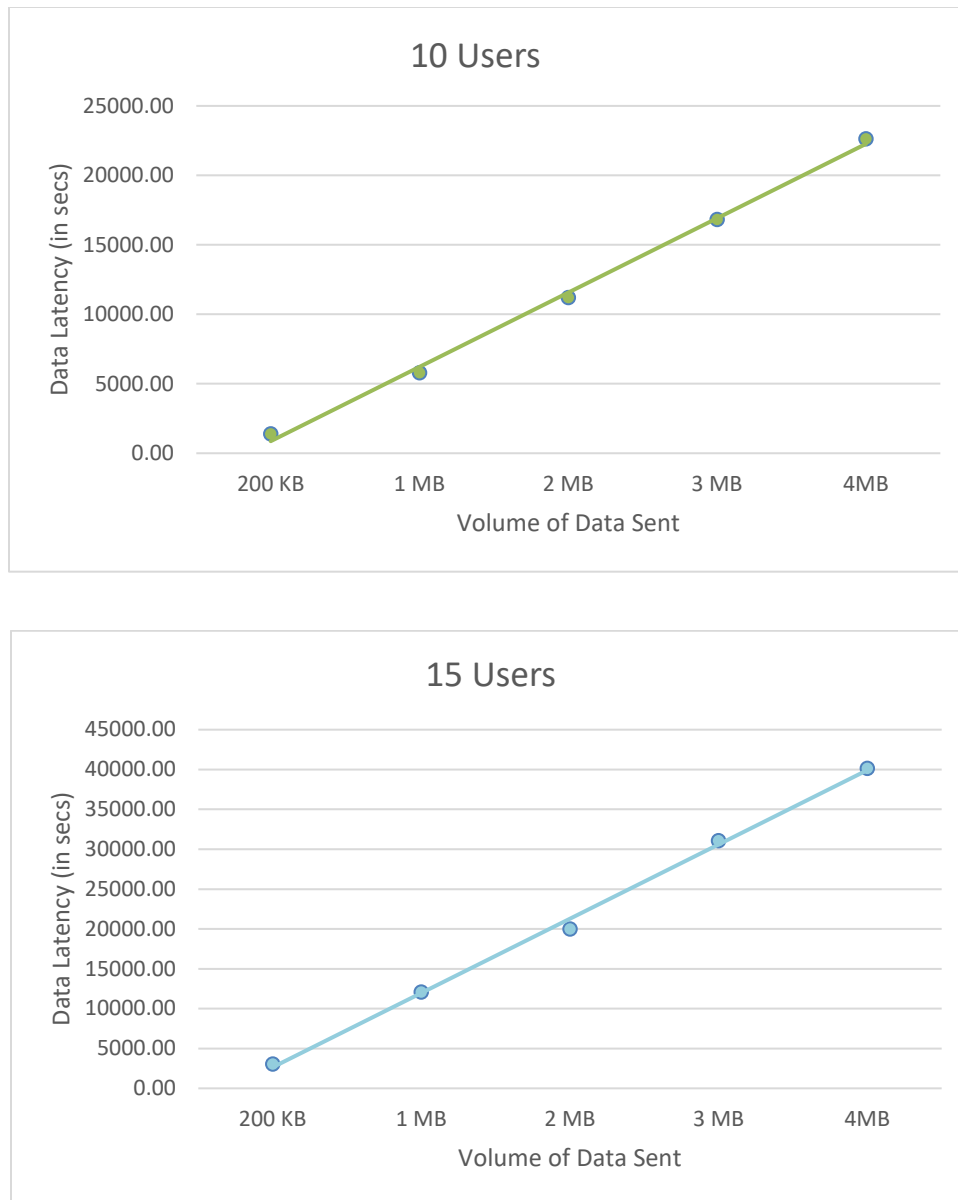


Fig. 12 (b). Database Latency vs Volume of Data Sent by Multiple Users

After observing the plots in Fig. 12. (b), we can roughly set a linear execution time in terms of database latency vs volume of data sent by multiple users. So we can see that as the number of users increases the time taken by multiple users for “text” insertions also increases.

## 6. YIOOP'S RECOMMENDATION SYSTEM

Before moving ahead and discussing about the potential enhancement to Yioop's recommendation system, we need to look at some of the concepts being used in the current implementation of this system. In particular we will look into term frequency and inverse document frequency (TF-IDF) in depth and word embeddings to add more detail to it.

When an information retrieval system like a search engine scores a document as relevant if it contains the terms in the user's search query it fails to take into account the number of occurrences of the query words in the document while weighing a document's relevance. Now, term frequency and inverse document frequency are designed to weigh the documents while taking into consideration the frequency of terms. A word's performance in TF-IDF is determined by how many documents it appears in compared to how often it appears in that document [5]. If a user query contains a word with a high TF-IDF value, the document would be interesting to the user if it contained this word.

### **6.1 TD-IDF Brief Synopsis**

The TF-IDF returns relevant documents according to the count of the words that are part of a user's search query. In order to get such documents two measures are calculated, i.e., term frequency and inverse document frequency. To better understand how this works we describe the inner workings of the technique and then work our way through an example. Assume we have a set of 'n' documents and user's query of 'm' words, we intend to return a smaller set of documents in order of most relevance based on the provided user query.

#### **6.1.1 Term Frequency (TF)**

The term frequency in documents refers to the number of times a word appears in a document. As an example, let's look at the three documents below and try to understand how the term frequency

calculation is done.

Document 1: Baguette a bread type can be made with the dry yeast or the fresh yeast.

Document 2: Toasted bread has a tasty pairing with the salted butter.

Document 3: You can make the beer from a dry yeast or a distiller yeast.

Let us say a user searches a query “bread pairing”, as such free-text queries are usually created by web users without following a particular language or structure. Instead, they use words to construct a query expression. There is no standard query structure that all users follow.

Table 1, 2 and 3 show the TF for all three documents respectively.

Table 1: Document 1 Term Frequency

words	baguette	a	bread	type	can	be	made	with	the	dry	yeast	or	fresh
frequency	1	1	1	1	1	1	1	1	2	1	2	1	1

Table 2: Document 2 Term Frequency

words	toasted	bread	has	a	tasty	pairing	with	the	salted	butter
frequency	1	1	1	1	1	1	1	1	1	1

Table 3: Document 3 Term Frequency

words	you	can	make	beer	from	a	dry	yeast	or	distiller
frequency	1	1	1	1	1	2	1	2	1	1

In a document the frequency of a word is influenced largely by the document's size, so we normalize it as a logarithm (log) of the frequency. In the case that a word appears only once in a document, that word has TF measurement of zero for that document. To avoid this, we add a constant 1.

Normalized TF is calculated as:

$$TF = \begin{cases} \log(f_{t,d}) & \text{if } (f_{t,d}) > 0 \\ \log(f_{t,d}) + 1 & \text{if } (f_{t,d}) = 0 \end{cases}$$

where TF = term frequency,  $f_{t,d}$  = frequency of a word 't' in document 'd'.

Table 4: Document 1 Log Term Frequency

words	baguette	a	bread	type	can	be	made	with	the	dry	yeast	or	fresh
frequency	1	1	1	1	1	1	1	1	0.3	1	0.3	1	1

Table 5: Document 2 Log Term Frequency

words	toasted	bread	has	a	tasty	pairing	with	the	salted	butter
frequency	1	1	1	1	1	1	1	1	1	1

Table 6: Document 3 Log Term Frequency

words	you	can	make	beer	from	a	dry	yeast	or	distiller
frequency	1	1	1	1	1	0.3	1	0.3	1	1

### **6.1.2 Inverse Document Frequency**

We consider all words in a document equally important when we calculate the term frequency. But, it overlooks the effect of a few words common to almost all documents. Some words like a, an, the, etc., are in almost all of the documents, while others are in only a few, in this situation, the logarithm is helpful.

IDF calculation:

$$IDF_t = \log\left(\frac{N}{N_t}\right)$$

where  $N$  = total documents in corpus and  $N_t$  = number of documents containing term 't'.

Let us look at how IDF is calculated for user's query "pairing",

Total document available in corpus ( $N$ ) = 3,

Number of documents containing term 't' ( $N_t$ ) = 1,

$$IDF_{pairing} = \log\left(\frac{N}{N_{pairing}}\right) = \log\left(\frac{3}{1}\right) = 0.48$$

IDFs calculated for all terms in our corpus are shown in Table 7.

Table 7 Inverse Document Frequency of all words in given corpus

words	IDF
baguette	0.48
a	0
bread	0.18
type	0.48
can	0.18
be	0.48
made	0.48
with	0.18

the	0
dry	0.18
yeast	0.18
or	0.18
fresh	0.48
toasted	0.48
has	0.48
tasty	0.48
pairing	0.48
salted	0.48
butter	0.48
you	0.48
make	0.48
beer	0.48
from	0.48
distiller	0.48

As we can see terms like “a”, “the” etc. have a low weight since they appear in most documents and words like “tasty”, “beer” etc. have a higher weight since they appear in just one document.

### **6.1.3 TF - IDF to Calculate Weights**

We have TF and IDF of words in given corpus, the next step is to multiply these two quantities to find out the frequently occurring words in a document and inseminate the influence of their frequency in the surrounding documents. The stop words such as is, an, etc. found in nearly all documents can be filtered out with this technique. By reducing the effect of stop words in the weighing process, this method allows you to gain a better understanding of the relevance of each word to a document.

So to find the relevant weights, we multiply each document's normalized term frequency with its inverse document frequency. Looking at our example further, in Doc. 1 the word “bread” has normalized

term frequency of 1 and IDF of 0.18 so the weight assigned to for that term is  $1 \times 0.18 = 0.18$ . Now, we find the TF-IDF weights for all documents and this is shown in Table 8.

Table 8 Documents Weighted by Query Terms

search terms	Doc. 1	Doc. 2	Doc. 3
bread	0.18	0.18	0.18
pairing	0	0.48	0

### **6.1.4 Cosine Similarity**

Using TF-IDF Weights, we can find the similarity between the user query and each of the documents. The cosine similarity is a measure of the importance of a document to a user. Just as we calculated the TF • IDF for the words in a document, we can calculate the TF • IDF for query terms. Based on cosine similarity (CS), we can determine whether or not a document  $D_1$  is relevant to a user query  $D_2$ . This is calculated using the following formula:

$$CS(D1, D2) = \left( \frac{D1 \cdot D2}{\|D1\| \cdot \|D2\|} \right)$$

$$where (D1 \cdot D2) = (D1[1] \cdot D2[1]) + (D1[2] \cdot D2[2]) + \dots + (D1[n] \cdot D2[n]),$$

$$\|Dn\| = \sqrt{(Dn[0]^2) + (Dn[1]^2) + \dots + (Dn[n]^2)}$$

The document with the highest cosine similarity measure to the query is returned because it is most similar to the query of the users.

### **6.2 Recommending Threads and Groups in Yioop**

Yioop initially would recommend threads using a baseline predictor typically implemented using a “rating” system, however since the rating system was not informative enough in Yioop, a user’s view of thread was used. This too ended up suggesting mostly the popular threads and so TD-IDF was introduced to improve the recommendations.

Initially, a Bag of Words (BoW) is created to using the “title” and “description” columns of the GROUP\_ITEM table shown in Fig. 1 of Chapter 3. Currently “Wiki” pages are excluded while looping over this table, moving ahead we will have to also exclude entries created for chats between users. After generating the BoW, the frequency count and log frequency of all those words is computed.

**6.2.1 Computing TF for Threads**

A BoW is created by iterating over each thread’s “title” and “description” as mentioned earlier and the log frequency for each word in the BoW is taken to reduce the impact of a large title or description in the table shown in Fig. 13.

ITEM ID	TERM ID	FREQUENCY	LOG FREQUENCY
443537	303434290	1	1
443537	405822459	1	1
443537	530372641	1	1
443537	1827795772	1	1
443537	735468453	1	1
443537	457581786	1	1

Fig. 13 ITEM\_TERM\_FREQUENCY Table

Here, ‘term\_id’ is generated using the ‘crc32’ hash value of the word in BoW in Yioop’s backend.

**6.2.2 Computing TF for Users**

A log of the user history is stored in the ITEM\_IMPRESSION table for each thread viewed by a user. The bag of words created in the earlier step is used to determine the importance of a word to each user. Using the ITEM\_TERM\_FREQUENCY table, we sum up the frequency counts for each word in a thread to determine how many times a user has seen the word. Next count of word occurrences that user has seen is stored using it’s log value in the USER\_TERM\_FREQUENCY table.



USER ID	TERM ID	FREQUENCY	LOG FREQUENCY
1	0	97	2.9867717342662
1	2966127	91	2.9590413923211
1	20262425	4	1.602059991328
1	53240003	91	2.9590413923211
1	62312701	95	2.9777236052888
1	66768310	91	2.9590413923211

Fig. 14 USER\_TERM\_FREQUENCY Table

### 6.2.3 Computing IDF for Threads

To get the IDF for each word in the bag of words, the number of times it appeared in each thread, versus the corpus of all threads as a whole is calculated. This was done using the ITEM\_TERM\_FREQUENCY table. The formula is as follows:

$$IDF_{w,t} = \log\left(\frac{\text{Total thread count}}{\text{Total threads containing word 'w'}}$$

where IDF 'w' = word with respect to thread 't'.

### 6.2.4 Computing IDF for Users

In a similar manner, the inverse document frequency for words with respect to users using the USER\_TERM\_FREQUENCY table shown in Fig. 14 is calculated. If there are words, i.e., threads, that are not being viewed by anyone, add 1 shown below.

$$IDF_{w,u} = \log\left(\frac{\text{Total users posts in Yioop}}{\text{Total users posts with the word 'w' + 1}}\right)$$

where IDF 'w' = word with respect to user 'u'.

### **6.2.5 TF-IDF weights for Threads and Users**

TF is multiplied by IDF for every word with respect to users and threads. The significance of a word to a thread is measured and stored in the ITEM\_TERM\_WEIGHTS Table as shown in Fig. 15.

TERM ID	ITEM ID	WEIGHT
1246902077	437176	0.95708015314937
1092272812	437176	3.2189420057495
1810991265	437176	1.5956149753564
1971925435	437150	4.6895811329413

Fig. 15 ITEM\_TERM\_WEIGHTS Table

Also, the significance of a word to a user is measured and stored in USER\_TERM\_WEIGHTS Table as shown in Fig. 16.

TERM ID	USER ID	WEIGHT
66768310	1	6.5538146239009
67305146	1	3.4410599427811
88636351	1	2.3462702187148
90542534	1	1.8169038393757

Fig. 16 USER\_TERM\_WEIGHTS Table

### **6.2.6 Thread and User Cosine Similarity**

Next, based on cosine similarity between users and threads, threads that are closest to each user's taste are determined. Finally, users are recommended the top three similar threads.

ITEM ID	USER ID	ITEM TYPE	SCORE	TIMESTAMP
429677	1327	2	0.98958075420778	1637691092
443364	1327	2	0.98907703090722	1637691092
443132	1327	2	0.95666169118361	1637691092
325	1	3	21.2678494472538	1637691092
324	1	3	17.8810669988863	1637691092

Fig. 17 ITEM\_RECOMMENDATION `Table

In Fig. 17 table, “item\_type” is used to distinguish between a thread and group recommendation, value 2 indicates it’s a thread and 3 indicates it’s a group.

### **6.3 Group Recommendations**

In addition to suggesting groups based on user interests, the system suggests groups that a user might be interested in and are not members of. Recommendations are made using thread titles and descriptions since the group names in Yioop are very generic and don't explain what the group is about. To calculate cosine similarity for each group, all the threads from the group are extracted and their cosine similarity is summed. Users are recommended the top three similar groups from the same table as in Fig. 17.

## 7. ENHANCING YIOOP'S RECOMMENDATION SYSTEM

So far, we have seen how the recommendation system in Yioop works and how TF-IDF is used to give user's recommendations that are closer to their tastes based on their thread viewing history. One thing to note is TD-IDF only considers a word's relevance in user query to a document and returns the most relevant documents based on the word from the entire available corpus. However, it fails to consider the user word in context to other words surrounding it. Thus, one way to enhance the currently established recommendation system would be to provide context to the words of interest in the entire corpus using the concept of word embeddings, particularly we will look at Hash2Vec.

### **7.1 Word Embeddings**

At its core, it is simply a method of associating words using vectors. The skip-gram model and CBOW mentioned earlier are mainly used to represent words as vectors. The Neural Network is required for both architectures in order to convert words into vectors. A CBOW and Skip-gram model are related to the context of the word, where CBOW attempts to predict the word based on the context, while a Skip-gram model attempts to predict the context based on the word. The initialization of word vectors is done by generating vectors with random real numbers, next by performing certain tasks, a program learns meaningful vectors. These words when represented in say a 2-Dimensional space based on their new calculated vectors should make intuitive sense. For example words like "placate" or "pacify" should appear together.

### **7.2 HASH2VEC**

To understand the working of Hash2Vec, we need to first look at hashing. Converting variable-length inputs into fixed-length outputs using some mathematical function, the process is known as hashing [10].

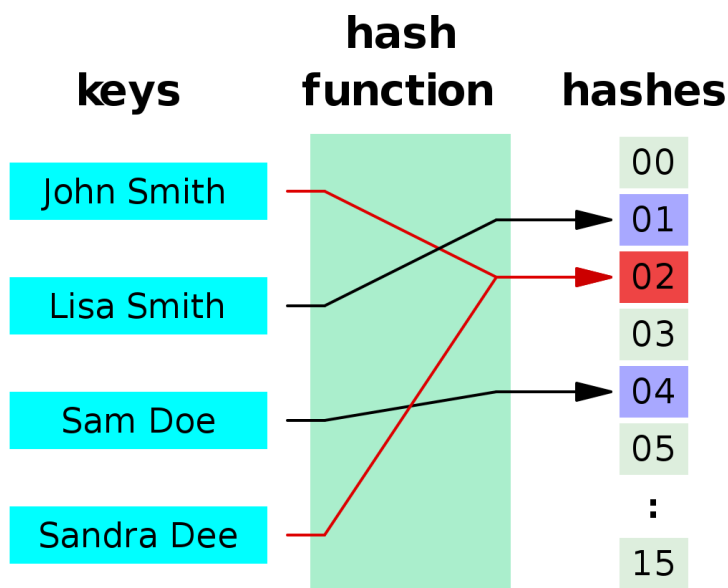


Fig. 18 Hashing [11]

An example of a hashing function for keys is shown in Figure 7, which maps them to integer values. As a mathematical function, a hash function processes input and converts it into a value that can be used. There are many applications for hashing, including security, optimization, and processing words. It has been around for a long time. Figure 7 illustrates that when two keys point to the same value, we have a collision. A good hash function minimizes such collisions and produces a result that fits in our table size. In order to solve the collision problem effectively, the hash function should run with a minimum computing time. Double hashing, linear or quadratic probing are all approaches to solving this collision problem [10].

When mapping keys to hash functions, the probability distribution should be as even as possible, so that uniform results would be produced. It is also possible to have the hash function generate results using an efficient table so that there are fewer collisions when it stores values and goes through them fast. This would result in better overall performance and less time spent going through values. The efficiency of a hash function depends on its performance and data storing capabilities. This is always a tradeoff. It is

important that a hash function produce the same outcome for the same key every time in larger applications to minimize collisions with larger storage spaces. In this project, we have used a vector size of 200 and used the first four bytes of md5 hash to map to it and resolve the collision problem as it gives us sufficient space.

Now coming to Hash2Vec, using a deterministic approach, Hash2vec creates vectors from words in a low-dimensional space. We will discuss in detail how these vectors are produced in the next section. This methodology was developed because the traditional method of creating vectors to represent each word in a low-dimensional space needed a lot of training when it was applied to neural networks. Using the Hash2Vec method, however, does not require any training, it merely attempts to derive a word hash from a context window. Using this method, the hash values of each word are stored a temporary storage system in the form of a dictionary or hash table. This process is called hashing with context, and it would take less time than word2vec training. When the same word appears in the corpus again, it updates its existing hash value.

The process of Hash2Vec in Yioop works as follows, we create our bag of words using the same the process as described in Chapter 5. Next, we create a tuple such that for every term in our BoW, we take 5 words before the term and 5 words after the term, here the value 5 is selected arbitrarily. Next we calculate the distance of the words from our 'term' of interest using the formula,  $(e^{-x})^2$ , where  $x =$  (position of word from 'term'/standard deviation of range  $(-n, n)$ ), here  $n = 5$ . The idea here is when calculating distance of word from 'term' we get a value between the range  $(0,1)$  and the closer the value to 1 the closer it's position is to the 'term' in the corpus.

Next, we calculate the hash value of the words to hash to the appropriate position in the vector of length 200 defined for each term in the BoW. The hash function takes the first 4 bytes of the md5 hash value of the word then we take the integer value of those 4 bytes. After that we do  $(\text{integer value} \% 200)$  which gives us the position to hash to in the  $\text{vector}_w[200]$ , where 'w' is part of the BoW. We then iterate over each newline in the corpus and do so for all words which we called as the 'term' of interest earlier.

Essentially the vector for each word in our BoW acts as a kind of definition for the word based on its context in a sentence. The different hash positions store its definition in different contexts.

Now, in order to find the most similar words we take the cosine similarity of our ‘term’ of interest vector and each word vector in the BoW. Then we filter out the words with the highest cosine similarity to the ‘term’ of interest. Now, we store this in a table called Hash2Vec as shown in Fig. 19.

TERM1	TERM2	SCORE
291958275	457101108	0.22062285087324
291958275	1425807786	0.2054091325479
291958275	1691756541	0.17786748510184
291958275	1387883559	0.07095626121466
291958275	12273120	0.04983362620488
291958275	1295930884	0.0425301615449
291958275	1738576411	0.02901137749091
291958275	345280726	0.00761595165231

Fig. 19 HASH2VEC Table

In Fig. 19, we see “Term1” refers to our “term” of interest stored as an integer, “Term2” are the words most similar to the “term” of interest using the Hash2Vec score. Now in the USER\_TERM\_WEIGHTS\_HASH2VEC table we update the TF-IDF weights by first multiplying the Hash2Vec score of the similar words and adding it to the original TF-IDF score, this is done for all the similar words user has seen, i.e., present in the table. We can see the cosine similarity changes from the original recommendation table vs the enhanced recommendation table shown in Fig. 20.

NEW_SCORE	OLD_SCORE	THREAD_ID	USER_ID
0.92783373765777	0.92741227739434	11047	1
0.92974029708099	0.92779276982726	392244	1

Fig. 20 Improved Scores for Recommendation Table

Finally the updated TD-IF scores are used to calculate the cosine similarity between user and thread and provides recommendations as is in the current system. After running the new recommender job on the Yioop database, we look closely at one of the results we see that recommendations do indeed change between the old and new. Further observing one of the results for a “user\_id” = 938, the old recommendation table gave the result in Fig. 21.

ITEM ID	USER ID	ITEM TYPE	SCORE	TIMESTAMP
429677	938	2	0.98958075420778	1638172189
443132	938	2	0.97023677469856	1638172189
317	938	3	26.548793645502	1638172189
325	938	3	21.5973811620012	1638172189
316	938	3	20.5289136839417	1638172189

Fig. 21 Result from Old Recommendation Table

And in the new recommendation table the result is in Fig. 22

ITEM ID	USER ID	ITEM TYPE	SCORE	TIMESTAMP
443486	938	2	0.99273694809558	1638172237
443370	938	2	0.99121369842996	1638172237
443429	938	2	0.98982944580506	1638172237
317	938	3	26.5713145938658	1638172237
325	938	3	21.6221855351121	1638172237
316	938	3	20.551912564768	1638172237

Fig. 22 Result from New Recommendation Table

If we look at the first recommendations between the tables in Fig. 21 and Fig. 22, they retrieve the threads titled “Happy New Year! August 2019 I did a couple 75 million page crawls .....” and “Post your solutions tot the Feb 17 In-Class Exercise to this thread. Best, Chris”. Now in order to judge if which of these recommendations is better for the user, we see the activity of the user. Since the user seems to be



actively contributing to the in-class assignment threads we can assume they are a student. We can also see that the first thread is a general update about the Yioop platform and the second thread is about an in-class exercise which the user may be more interested in. On observing this thread we see that words like “post”, “in-class” etc. all have the word “solution” as a similar word, hence the context seems to be preserved as intended and provides relevant thread recommendations.

### **7.3 Experiments**

To judge the accuracy of the hash2vec implemented recommendation system we use precision and recall. Precision for the first ‘k’ results is given by,

$$\frac{| \text{Rel} \cap \text{Res}[1..k] |}{| \text{Res}[1..k] |}$$

where Rel = is all the relevant documents in this case ‘threads’ and Res = the total thread count returned by the recommendation system. Recall for first ‘k’ results is given by,

$$\frac{| \text{Rel} \cap \text{Res}[1..k] |}{| \text{Rel} |}$$

We observed the results for 10 users both in the current recommendation system and the hash2vec implemented system and state the result below in Table 9.

	Current Recommendation System		Hash2Vec Recommendation System	
	precision	recall	precision	recall
Student	0.60	0.025641	0.83	0.042735
Admin	1.00	0.000123	1.00	0.000123
User	1.00	0.000354	1.00	0.000531
Student	0.67	0.000354	0.67	0.000531
Student	0.67	0.000354	1.00	0.000354
Student	0.67	0.000354	1.00	0.000354
Student	0.67	0.000354	1.00	0.000354
User	0.50	0.000266	0.33	0.000177
Student	0.83	0.000443	0.83	0.000443
Student	0.67	0.000443	0.83	0.000443

Table 9 Precision and Recall Results

Here, we can see that the hash2vec implemented recommendation system has at least the same precision and recall as the current recommendation system and in some instances gives preforms higher precision and recall. The current recommender system has an avg. F1 measure of 0.005714 and the Hash2Vec system has a measure of 0.009159, showing an increase of 0.003444 or 60. 28%. Additionally, we noted that since Yioop is configured to recommend the top three most similar threads and groups to users for some of the users the current recommendation system could not satisfy that criteria and showed fewer suggestions however, our hash2vec recommender for those same users could meet that criteria and provide more suggestions.

## 8. CONCLUSION

We researched some architectures and protocols that could work for direct messaging. We studied in depth the internal working of Yioop to determine the tables that we are of interest to us to be able us to develop the DM system. For old and new users, we developed a "Personal" group in Yioop to facilitate quick communication. We used AJAX to interact with the database and fetch messages instantly. The experiments we performed shows the database latency vs volume of data inserted by multiple users increases linearly based on the chi-squared test. A possible future work for DM system is improving the latency with multithreading to control multiple user insertions or adding a group function between three or more users to allow a multi-way private communication between users.

We studied Yioop's current recommendation system that suggests threads and groups which may be of interest to users using the user's viewing history and engagements in Yioop. Next, we implemented a Hash2Vec that uses the similarity between words to improve the recommender system in Yioop. Based on the experiments we performed on the Hash2Vec system we see an improvement of 60.28% in the avg. F1 measure and we can observe that the performance is on par with the current recommendation system in Yioop or in some instances Hash2Vec performs better by either giving higher accuracy or more recommendations.

## BIBLIOGRAPHY

- [1] “Seek Quarry”, Retrieved November 26, 2021, Available at: <https://www.seekquarry.com/>.
- [2] “Yioop”, Retrieved November 26, 2021, Available at: <https://www.yioop.com/>.
- [3] Akinbi, A., Ojie, E. Forensic analysis of open-source XMPP/Jabber multi-client instant messaging apps on Android smartphones. *SN Appl. Sci.* 3, 430 (2021), <https://doi.org/10.1007/s42452-021-04431-9>.
- [4] Carlos A. Gomez-Uribe and Neil Hunt. 2016. The Netflix Recommender System: Algorithms, Business Value, and Innovation. *ACM Trans. Manage. Inf. Syst.* 6, 4, Article 13 (January 2016), 19 pages. DOI:<https://doi.org/10.1145/2843948>.
- [5] Ramos, J, "Using TF-IDF to Determine Word Relevance in Document Queries.", Available: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.121.1424&rep=rep1&type=pdf>
- [6] R.B. Jennings, E.M. Nahum, D.P. Olshefski, D. Saha, S. Zon-Yin, C. Waters, "A Study of Internet Instant Messaging and Chat Protocols.", *IEEE Network* 20.4, 2006, pp. 16-21, doi: 10.1109/MNET.2006.1668399.
- [7] Mikolov, T., Yih, W. T., Zweig, G.: Linguistic Regularities in Continuous Space Word Representations. In *HLT-NAACL*, 746–751 (2013).
- [8] ] Luis Argerich, Matias J. Cano, and Joaquin Torre Zaffaroni: Hash2Vec: Feature Hashing for Word Embeddings(2016).
- [9] Nikolaj Cholakov. 2008. On some drawbacks of the PHP platform. In *Proceedings of the 9th International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing (CompSysTech '08)*. Association for Computing Machinery, New York, NY, USA, Article 12, II.7–2. DOI:<https://doi-org.libaccess.sjlibrary.org/10.1145/1500879.1500894>.
- [10] “Translate Microsoft”, Retrieved November 26, 2021, Available: <https://www.microsoft.com/en-us/translator/>.
- [11] “Hash Function” , Retrieved November 26, 2021, Available: [https://en.wikipedia.org/wiki/Hash\\_function](https://en.wikipedia.org/wiki/Hash_function)

- [12] Saksham Sachdev, Hongyu Li, Sifei Luan, Seohyun Kim, Koushik Sen, and Satish Chandra. 2018. Retrieval on source code: a neural code search. In Proceedings of the 2nd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages (MAPL 2018). Association for Computing Machinery, New York, NY, USA, 31–41. DOI:<https://doi.org/10.1145/3211346.3211353>
- [13] Donald Metzler, Yi Tay, Dara Bahri, and Marc Najork. 2021. Rethinking search: making domain experts out of dilettantes. SIGIR Forum 55, 1, Article 13 (June 2021), 27 pages. DOI:<https://doi.org/10.1145/3476415.3476428>.
- [14] Sarika Padmashali, “An Open Source Discussion Group Recommendation System”, San Jose State University, May 2017.