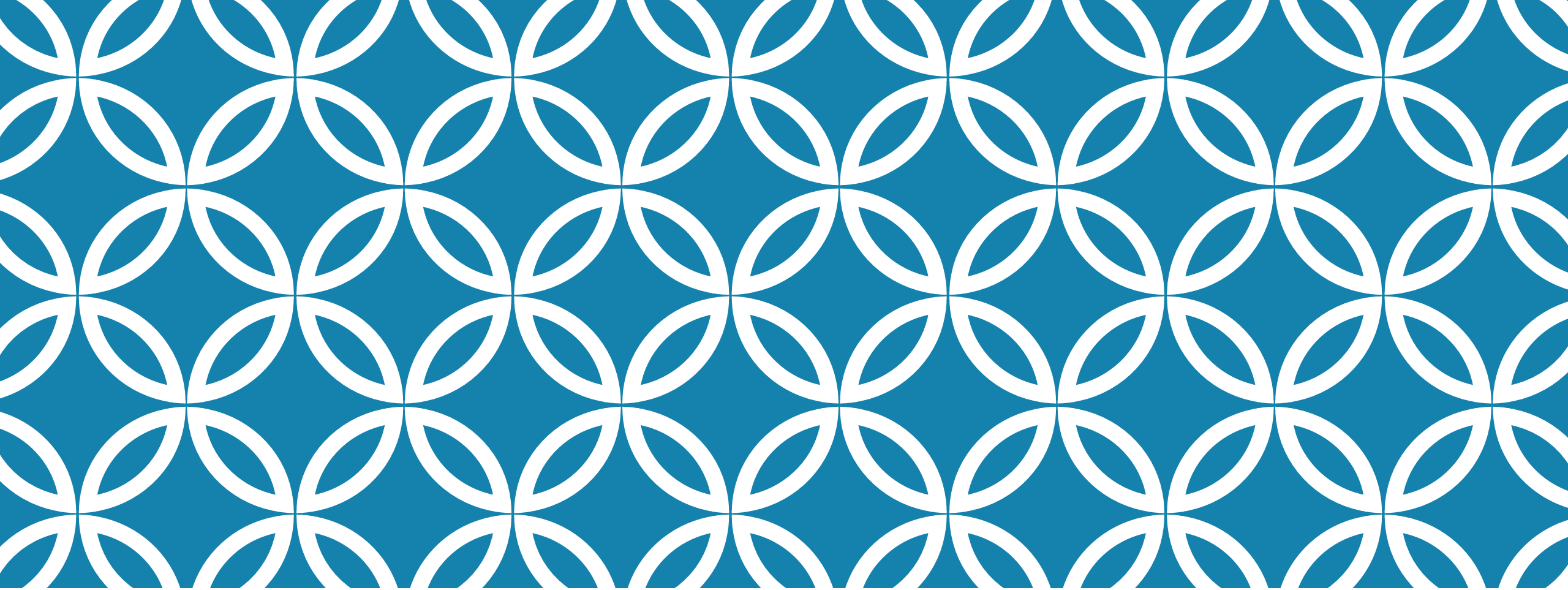


AN OPEN-SOURCE DIRECT MESSAGING AND ENHANCED RECOMMENDATION SYSTEM FOR YIOOP

BY
ANIRUDDHA MALLYA

TABLE OF CONTENTS

PURPOSE

- Yioop is an open-source implementation that acts as a search engine and web portal.
- As a web portal it lacks features like Direct messaging (DM), in this project we add this feature to Yioop.
- Yioop also uses a recommendation system that uses Term Frequency – Inverse Document Frequency which makes use of user's viewing history to recommend relevant threads and groups.
- We further extend this functionality by using Hash2Vec to improve the recommendation in Yioop.

INTRODUCTION.

- Topics of discussion:
  1. Direct Messaging (DM)
  2. Recommendation System

- DM is a type of technology that allows one to chat online with other users in real time over any type of computer network like the Internet.

- In a recommendation system, users are given suggestions as to which news articles to browse, which movies to watch, etc. so we can potentially find the information most relevant to us with little effort.

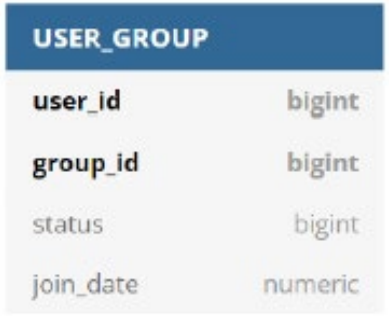
- Yioop makes use of a such a recommender system.

HISTORY OF DM.

- In the 80s, Internet relay chat allowed users to connect to networks with client software to chat with groups in real time.
- In the 90s, AOL messenger used the Oscar protocol and was the first to introduce the Buddy List system and Yahoo Messenger used the YMSG protocol.
- Facebook's Messenger and WhatsApp both use the XMPP protocol both follow the client server architecture. Both use end to end encryption however messages sent in WhatsApp are transient.
- XMPP is widely used as an instant messaging protocol, and it uses bidirectional streams over synchronous http (BOSH), we instead went ahead with an AJAX style implementation which can support long polling, the primary feature of BOSH.

DESIGN FOR DIRECT MESSAGING

- The following tables: USERS, USER_GROUP, GROUP_ITEM and SOCIAL_GROUPS were deemed relevant for implementing the DM system in Yioop.

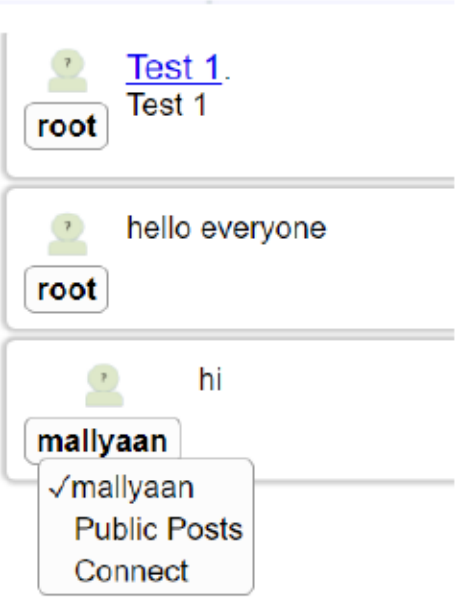


DESIGN FOR DIRECT MESSAGING

- When a new user is introduced into the Yioop environment and that user logs in for the first time a Personal group is created.
- The SOCIAL_GROUPS table manages the group information for a particular user.
- Allow users to connect with other users through a drop-down option.

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

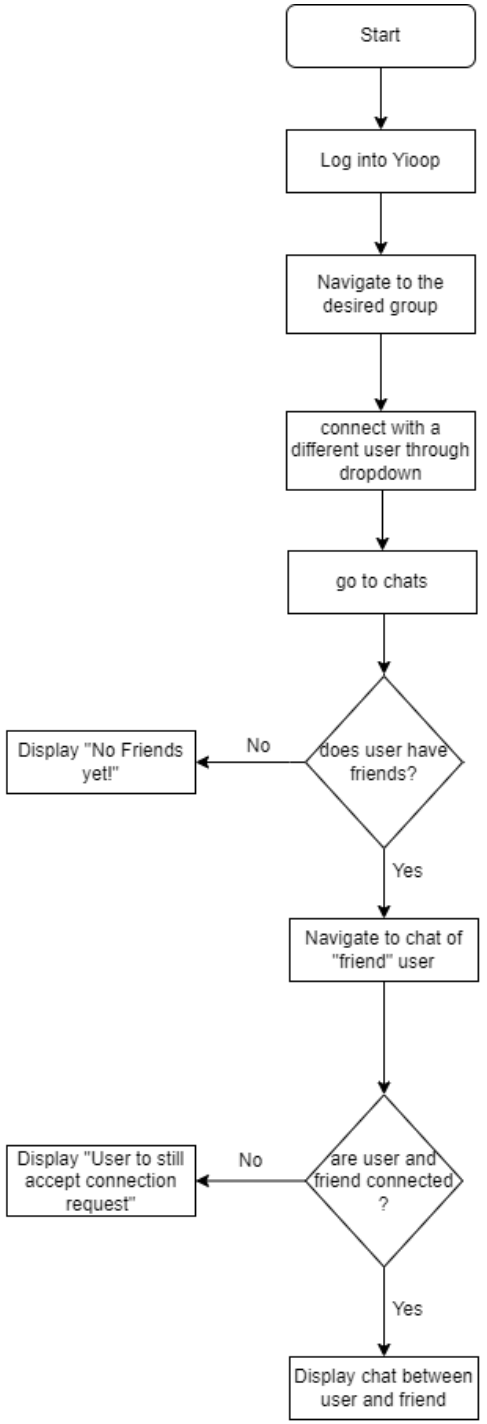
SOCIAL_GROUPS Table



IMPLEMENTATION FOR DIRECT MESSAGING

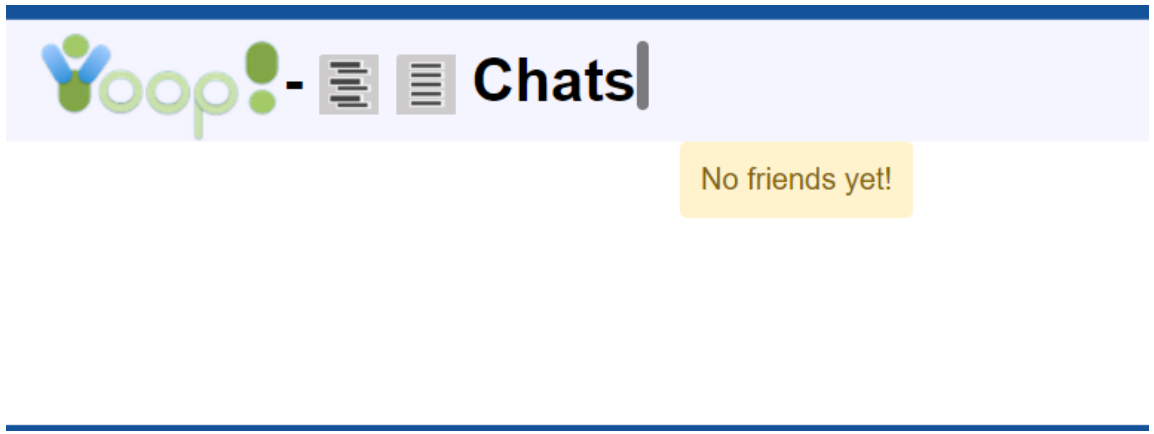
- There are three uses cases for this problem statement , i.e., handling the logic for when a user has no friends, when one user sends a friend request while the other user has not accepted the connection request and finally when both users have accepted the connection requests from each other.

- Yioop follows the Model-View-Controller (MVC) model all the logic must be handled by the controller.



USE CASE 1

- When a user has no connections , i.e., no friends.
- We do this by checking if a user has any friends—equated to threads—as part of their “Personal” group.



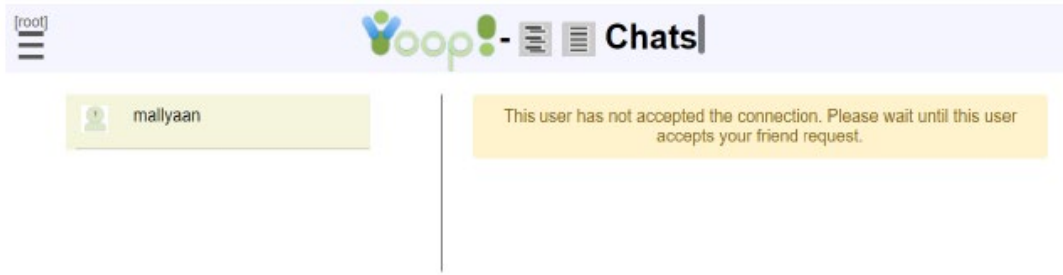
USE CASE 2.

- When a user sends the connection request to a different user2 and the connection has not connected with user2 then the connection is handled by prompting the user to wait for the connection to connect with user2.

- First the user gives the connection access to their "Personal" group, this handled in the backend database using the USER_GROUP table.

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

USER_GROUP Table



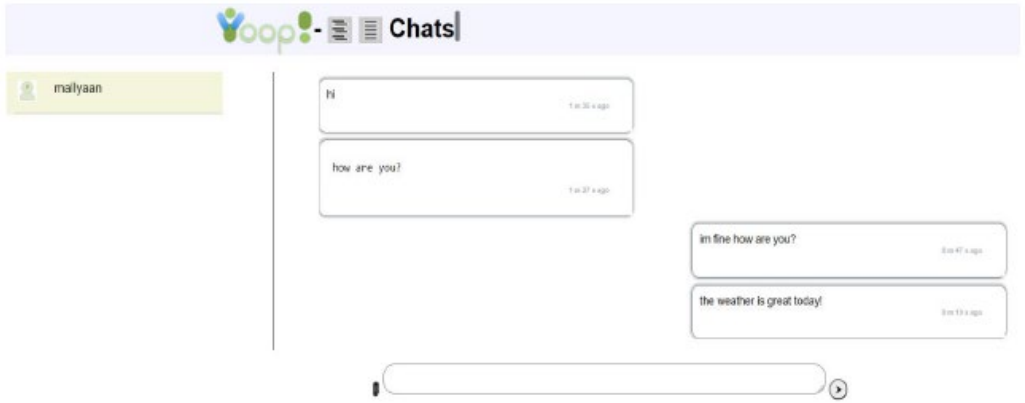
USE CASE 3

- Final use case three, when both users are connected to each other which is indicated by the USER_GROUP table as shown earlier, we then must store the chat between any two users, to do so we user the GROUP_ITEM table.

- 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

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

GROUP_ITEM Table

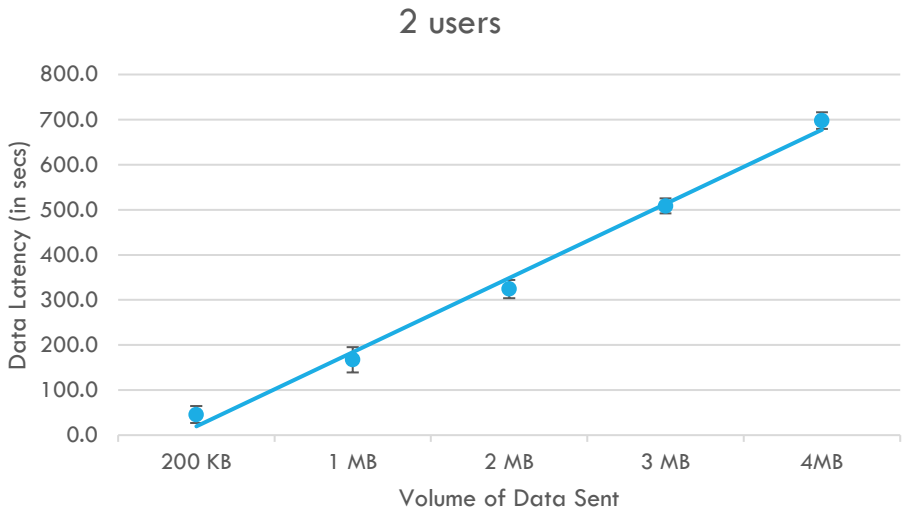


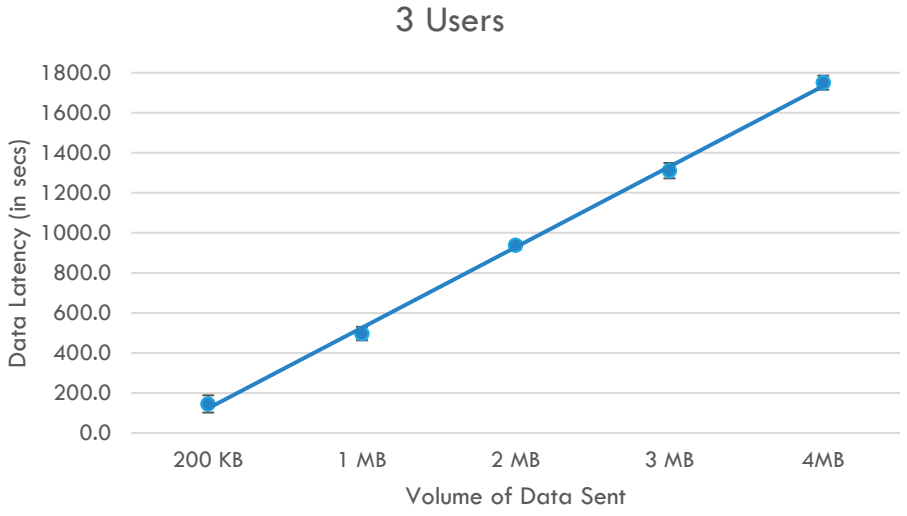
EXPERIMENTS FOR DM

- 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.
- 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 on the same local machine.

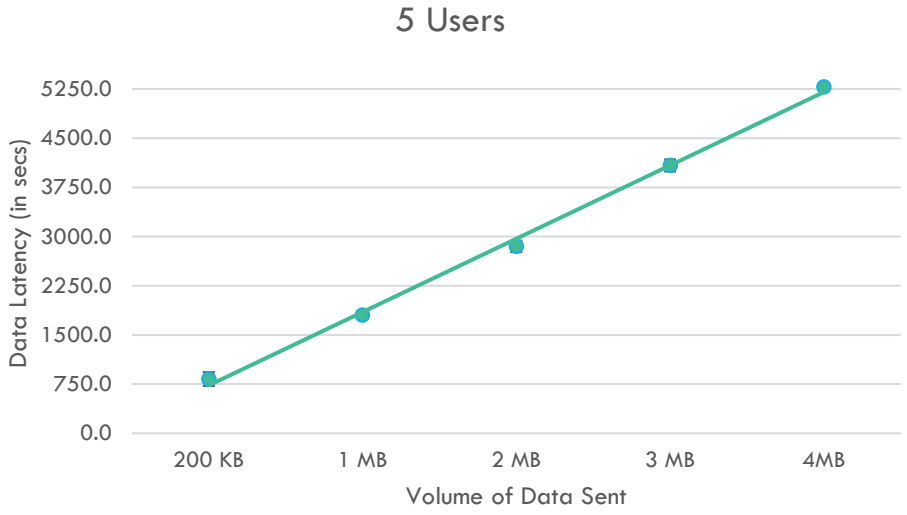
EXPERIMENTS FOR DM.

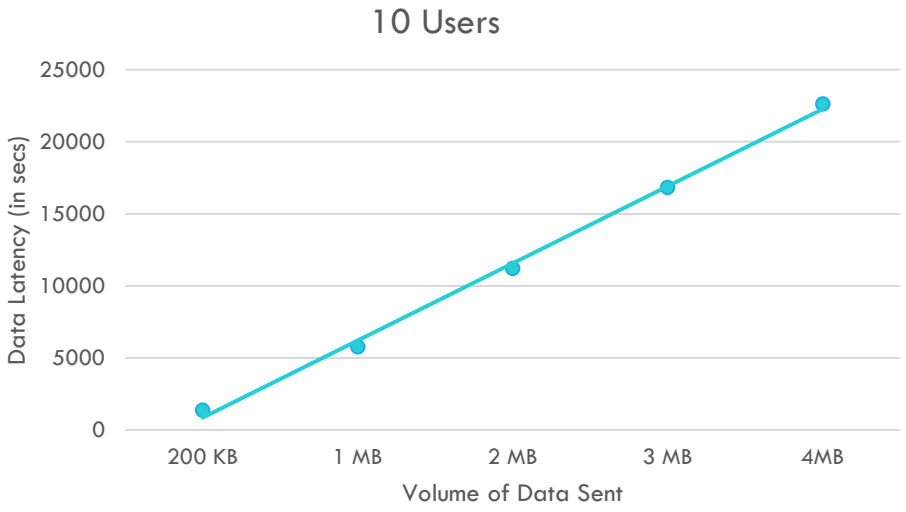
- The error bar is 3 standard deviations from the mean.
- We can see that as the number of users increases the time taken by multiple users for “text” insertions also increases.

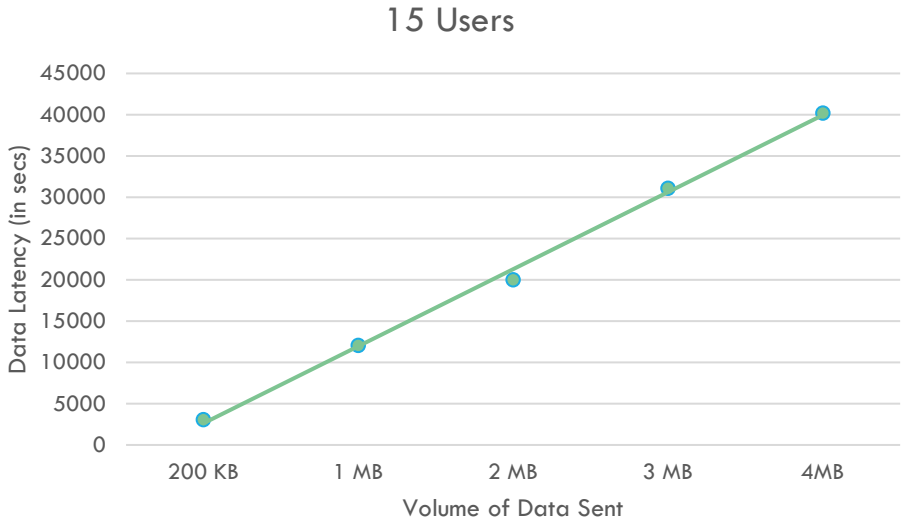




EXPERIMENTS FOR DM







CHALLENGES WITH DM.

- The first experiment we tried was creating a "Personal" group in one click for all existing users, but, since there were several different tables that needed to be changed, this proved to be challenging to handle.
- 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 using SOCIAL_GROUPS table, such as all the groups they are a part of or the menu bar.
- The third challenge was to get two different web pages to display on the same page, since it’s based on the existing group display functionality, we had to essentially combine the design for the two separate pages into one page.

HISTORY OF RECOMMENDATION SYSTEMS.

- The basics of recommender systems were founded by researches into cognition science and information retrieval, and its first manifestation was the Usenet communication system created by Duke University in the second half of the 1970s.
- The first known such solution was the computer librarian Grundy, which first interviewed users about their preferences and then recommended books to them considering this information.
- Since then two very different directions of recommender systems have evolved over time: collaborative filtering and content-based filtering.
- The former attempts to map (profile) the taste of users and offers content to them that users with similar preferences liked.
- The content-based filtering is about knowing the dimensions of the entity to be recommended.

YIOOP'S RECOMMENDATION SYSTEM.

- 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 consider 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.

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 assume that a user has entered a query q: bread pairing. We can sample Term frequency Table for Document 3.

Normalized TF is calculated as:

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

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

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

INVERSE DOCUMENT FREQUENCY (IDF)

- 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 the documents, while others are in only a few, in this situation, the logarithm is helpful.
- 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 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'.

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

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

Sample IDF Table

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.

- Looking at our example below, 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 x 0.18 = 0.18.

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

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.
- Formula used,

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

$$where\ (D1 \bullet D2) = (D1[1] \bullet D2[1]) + (D1[2] \bullet D2[2]) + \ldots + (D1[n] \bullet D2[n]),$$

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

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/voting system was not informative enough in Yioop, a user’s view of thread was used.
- This ended up suggesting mostly the popular threads and so TD-IDF was introduced to improve the recommendations.
- Currently “Wiki” pages are excluded, moving ahead we will have to also exclude entries created for chats between users in the GROUP_ITEM Table.

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.
- Here, 'term_id" is generated using the 'crc32' hash value of the word.

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

ITEM_TERM_FREQUENCY Table

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 count for each word in different threads 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

USER_TERM_FREQUENCY Table

IDF FOR THREADS & USERS

- 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 is calculated. This was done using the ITEM_TERM_FREQUENCY table. The formula is as follows:

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

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

- The inverse document frequency for words with respect to users using the USER_TERM_FREQUENCY table is calculated. If there are words, that are not being viewed by anyone, add 1.

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

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

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.
- Also, the significance of a word to a user is measured and stored in USER_TERM_WEIGHTS Table.

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

USER_TERM_WEIGHTS Table

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

ITEM_TERM_WEIGHTS Table

THREAD AND USER COSINE SIMILARITY.

- 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_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.

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

ITEM_RECOMMENDATION Table

GROUP RECOMMENDATIONS

- In addition to suggesting threads based on user interests, the Yioops recommender also suggests groups that a user might be interested in and are not members off.
- 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.
- Users are recommended the top three similar groups as shown in the table to the right.

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

ITEM_RECOMMENDATION Table

ENHANCING YIOOP'S RECOMMENDATION SYSTEM

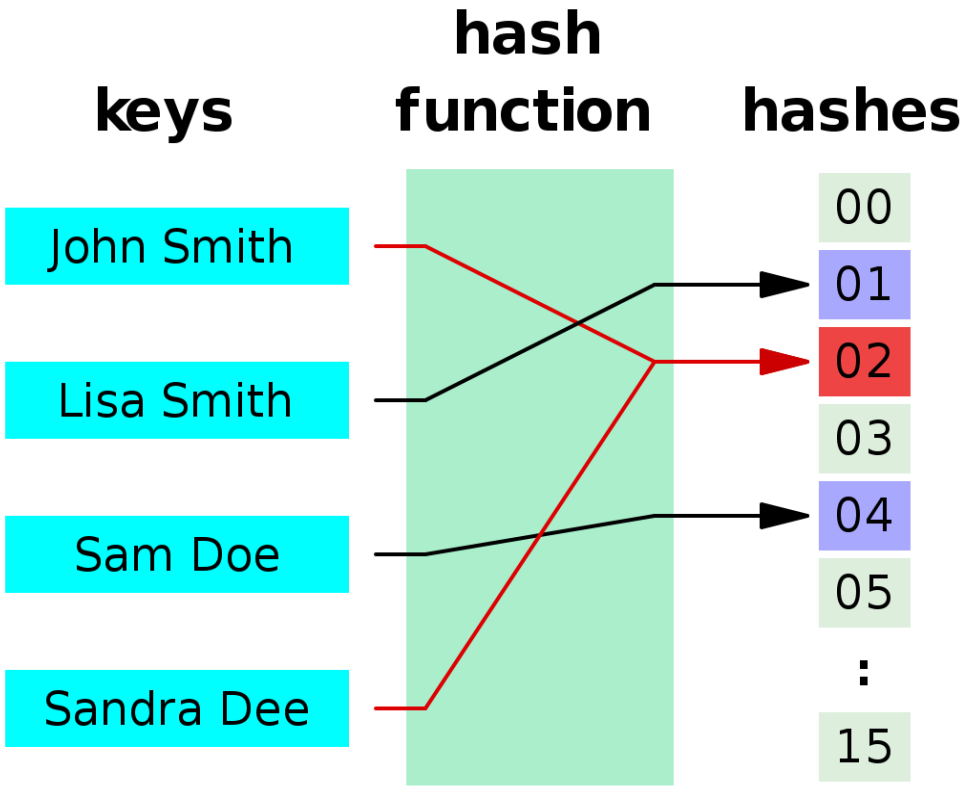
- We have seen how the recommendation system in Yioop works and how TF-IDF is used to give user's recommendations that are closer to their tastes based on their thread viewing history.
- TD-IDF only considers a word's relevance in user query to a document and returns the most relevant documents based on the word from the entire available corpus.
- However, it fails to consider the "user word" in context to other words surroundings it.
- 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.

WORD EMBEDDINGS

- At its core, it is simply a method of associating words using vectors.
- The skip-gram model and Continuous Bag of Words (CBOW) are mainly used to represent words as vectors in Neural models.
- However, say, we use a 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.
- We decided to try a Hash2Vec model that does create vectors in a non-neural way, i.e., without any training models but instead uses a hashing technique and has a space complexity of O(nk), n= number of words and k = some fixed dimensionality and can be small.

HASH2VEC.

- When converting a variable-length inputs into fixed-length outputs using some mathematical function, the process is known as hashing.
- As a mathematical function, a hash function processes input and converts it into a value that can be used.
- A good hash function minimizes 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.



HASH2VEC.

- Using a deterministic approach, Hash2vec creates vectors from words in a low-dimensional space.
- 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. This process is called hashing with context.
- When the same word appears in the corpus again, it updates its existing hash value.

DESIGN OF HASH2VEC

- 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.

- We then 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) as vectors are normalized and the closer the value to 1 the closer it's position is to the 'term' in the corpus.

- 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.

IMPLEMENTATION OF HASH2VEC

- 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.
- 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

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

HASH2VEC Table

HASH2VEC TABLE

- We see “Term1” refers to our “term” of interest stored as an integer, “Term2” are the words most like 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 on top.
- We can see the cosine similarity changes from the original recommendation table vs the enhanced recommendation table in the figure at the bottom.

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

HASH2VEC Table

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

COMPARING OLD VS NEW RECOMMENDER SYSTEM

- Looking at the recommendations between the tables, 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”.

- We can 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.

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

Old recommendation system

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

New recommendation system

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,

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

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,

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

We observed the results for 10 users both in the current recommendation system and the hash2vec implemented system.

EXPERIMENTS

- 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 0.005714825 and the Hash2Vec system has a measure of 0.00915971, showing an increase of 0.003444885 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.

	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

CONCLUSION

- We studied the internal working of Yioop to determine the tables that we are of interest to us to be enable 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 sent by multiple users increases roughly linearly in time.

CONCLUSION

- 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 power with the current recommendation system in Yioop or in some instances Hash2Vec performs better by either giving higher accuracy or more recommendations.

REFERENCES

[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=10.1.1.121.1424andrep=rep1andtype=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.

REFERENCES

[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

[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] Gaikwad, N, “Compare word2vec with hash2vec for Word Sense Disambiguation on Wikipedia Corpus”, San Jose State University, May 2020.