ISSN 1330-3651 (Print), ISSN 1848-6339 (Online) https://doi.org/10.17559/TV-20230511000622
Original scientific paper

FBF: A High-Efficient Query Mechanism for Keyword Search in Online Social Networks

Jinzhou HUANG, Yan TONG, Bo HANG, Degang XU, Feng WANG, Jing YU*

Abstract: The widespread adoption of online social networks has facilitated content sharing among users. However, privacy controls restrict users' access to only a limited portion of the network, typically limited to direct connections or two-hop friends. Browsing relevant profiles and home pages has become a common practice for users, but the vast amount of data involved often hampers their ability to efficiently retrieve the desired information. This paper presents an efficient keyword search model designed to aid users in accessing the required information effectively. Leveraging advancements in Bloom filter technology, we propose a novel summary index called Friend-based Bloom filter (FBF), which enables large-scale full-text retrieval while reducing inter-server communication costs and query latency. We conduct a comprehensive simulation to evaluate our ranking model, and the results demonstrate the effectiveness of the FBF scheme. Specifically, our approach achieves a reduction of 92.4% in inter-server communication costs and 78.7% in query latency, with a high-search precision condition resulting in a remarkable 98.3% improvement.

Keywords: friend-based Bloom filter; keyword search; online social network

1 INTRODUCTION

Distinct from web-based information sharing and search engines, the characterristic for OSN is that most of the users' behaviors are navigating the dynamics and day-to-day life of the circle of friends [1]. One study found that nearly 92% of Facebook users' behavior was based on browsing their friends' profiles, while in traditional web search users publish their data for sharing [4]. OSN users, on the other hand, pay much more attention to the personal privacy issues. Facebook, for example, limits the user privacy control by providing a user with the privacy policy setting with four options "everyone", "friends of friends", "friends only" or "only me". One study shows that a fraction of 63% of Facebook users set the policy to only allow access by friends within a distance of two hops [5]. Based on an extensive topology trace we gathered from Facebook, on average, a user has 3.1×10^4 two-hop friends, and more than 40% of them have over 1.0×10^4 friends or friends of friends [6]. Since there may be a large number of relevant datasheets, a simple exploration operation cannot efficiently obtain the information for a user's needs. The approach to large-scale social searching is to gather relevant information together and construct a globle search index [7]. Due to the aforementioned user privacy concerns, it is difficult to construct a keyword search for OSN. For this reason, a novel distributed keyword search scheme for online social networks is proposed in this paper. Facebook, for instance, uses Cassandra [8] as a backend storage system, which is a key-value distributed mechanism, for a variety of services, such as inbox search. Cassandra, in particular, uses a ring infrastructure and uniform hashes with the goal of providing high scalability. A consistent hash function uses a basic hash function such as SHA-1 to assign an m bit identifier to the user ID and the certain server in the Cassandra system. To accomplish this, a user's data is placed on the first server with a hash value that either equals or follows the ID of the user being hashed. Since each server handles nodes on a DHT ring, it can only affect its near neighbors on the ring, while the others do not. However, existing OSNs struggle to provide keyword-based content searching, and often use the key-value mechanism of DHT to serve as a simple search for content based on username [8]. The random partitioning strategy of the hash function is essentially the same, e. g., users' friends are randomly assigned to servers via data centers. For a user's two-hop friends, the simple query processing usually consists of communication across a network of data centers, leading to an increase in the amount of data center communication overhead, which is especially significant in high-load data center networks [9]. To solve this issue, we propose a novel keyword search scheme through the use of a variant of Bloom filter [10], which is also widely used in hash technology [11]. The mechanism we used constructs a small summary index with the related friends. A ranking model is proposed based on the summary index to determine which server is more likely to return the predicted result of a query, rather than transmitting a query message to the whole network for relevant answers. In this scheme, unnecessary server accesses can be checked and filtered, only query messages can be transferred to the top server, avoiding a large number of unnecessary messages due to exhausting search, and to significantly reduce the cost of communication during query processing. Our model takes into account two factors: 1) Using concise local summaries, for each user, the TF × IDF score can be estimated for the accuracy of responses to a particular query. 2) Using a novel Bloom filter to support the temporal factor of users' data and a ranking model for the top k responses for a certain query. We denote the novel Bloom filter as Friend-based Bloom Filter (FBF). FBF not only supports the representation of dynamic ensembles, but it can also incorporate temporal information into the summary index for time-intensive classification. In order to evaluate the performance of the design, we performed a simulation using the test collection from the TREC WT10G [14], as well as query logs from commercial search engines [15]. To summarize, the results of our experiments show that our FBF scheme can achieve high query quality, as well as significantly reduce communication overheads. In summary, the main contribution of this paper is two-fold.

(1) We propose an efficient summary index by using the Friend-based Bloom filter (FBF), and construct a novel ranking model for top *k* responses of a query to avoid unnecessary access to a large number of servers.

(2) To evaluate our model, we performed a simulation to demonstrate that our design is effective.

2 LITERATURE REVIEW

An OSN search engine should implement keyword-based content search as well as enforcing access control [16]. Using Key-Value stores, legacy OSN systems provide a name based lookup service. Facebook, for example, has the "friends search" service based on user names and the "inbox search" service that allows users to search by using names as the queries [8]. Orkut users are able to add additional restrictions to assist the user in filtering results with serval coefficients [17]. By integrating social networking with public web search to supplement search results, the complementary search attempts to compensate for the dissatisfying user experiences of OSN searching [18]. Although the complementary search provides users with more results from public Web, the increasing large scale data accumulated inside social networks remains unsearchable. More recently, keyword searching in the key-value model has attracted a great deal of research attention due to its widespread use in web search services. Distinct from the traditional entity-relationship model, the key-value model stores the entity as key-value pairs. It keeps each entity in a eqle, and it also partitions tables to many servers in order to achieve a fast response. In OSN's search for keywords is more difficult [19], the information recency plays important non-user ranking roles [20]. In order to address content extraction in OSNs, this paper proposes a novel ranking model that takes into account two major factors: content relevance and data recency. We design a summary index by extending Bloom filter techniques in order to realize the ranking model. Microblogging is another kind of OSN, and the microblogging content search has also attracted many communities to research [21]. It focuses on novelty and popularity of contents, such as Twitter, which is both a news media and a social network, without considering access control policy [24]. So, the microblogging search is very different from the OSN search we are considering. On the basis of the Key-Value model, Arash et al. [26] proposed the KSTORE scheme to rank the relevance of tables with a certain query, which mainly considers the structure information of tables and summarizes this structure information by using entropy theory for future query evaluation. Therefore, the Microblogging content search is very different from the OSN keyword search discussed in this article. Another style of OSN search service is the question/answer (Q/A) over OSNs [27]. Horowitz et al. [28] designed Aardvark, where users can ask questions via IM, email, text or voice. Aardvark then sends the question to anyone who might be able to answer it on the extended social network. While their experimental results showed that half of the questions were answered for the first time in 10 minutes, this efficiency may not be enough for online users. Another problem with Aardvark is that it maintains a centralized index for query routing. Unlike Aardvark, our design solves the problem of searching for content in the growing trove of data accumulated on the social networks.

3 RESEARCH METHODOLOGY

This section presents the design of the keyword search scheme on OSNs. This section first provides a brief overview of the design, and then focuses on the design of the summary index by taking advantage of Friend-based Bloom filters. Next, we introduce the ranking model to select top-k servers and eventually rank the results.

3.1 Solution Outline

Within the system, each user of the OSN maintains a simple summary index of which friends that can be contacted. In particular, each user caches an index table for summaring the content of friends. Such a summary index contains a Friend based Bloom filter. The FBF summarizes a set of terms extracted from the corpus along with the temporal information extracted. The FBF is a variant of the Dynamic Bloom filter [29]. Upon arrival of a query process, we first perform a local search through the user's summary index table and filter out those friends who do not have the answers based on their FBF scores. In this scheme, the matched documents of relevant friends are retrieved and the FBF-based summary index forms the heart of the design.

3.2 FBF Based Summary Index

As discussed above, the issue of privacy complicates the construction of a centralized index for social search. Rather, the scheme maintains a local summary index of a small portion of the users. For a given user, it is easy to summarize the contents of friends and to compute the index of all friend sets in advance. The use of this summary index means that when a query arrives, it forwards the query only to friends who might respond to a certain query, thus avoiding a large amount of bandwidth overhead caused by unnessary accesses. Then, the goal of user index design should satisfy serval aspects: 1) It should be space-efficient due to the large size of the OSN population. A succinct summary index can be loaded into memory when needed, thus the lookups and scans can be implemented without touching the disk. 2) It should have enough space to support the dynmic appending because the content of users can be appended over time. 3) It suggests that the summary should contain item recency information because of a great deal of interesting information recency information for social behaviors [19]. It is well known that Bloom filter [6] is a random data structure for the concise representation of an ensemble, which is used extensively in online social networks [32]. A standard Bloom filter (SBF) is basically a vector, which makes it easy to test membership. We referred to the schema as the schema Dynamic Bloom Filter (DBF). In the first stage, a DBF has a single active SBF. When the number of elements inserted in to SBF goes up to n_0, which causes the SBF's false positive rate to be greater than the acceptable threshold ε, the SBF is named full and a new SBF is appended to the DBF. When the DBF adapts to a new item, it inserts it into the newly appended active SBF. The false-positive rate of DBF is the probability that at least one SBF has been falsely paired. The false positive rate can be calculated as follows:

$$f_{DBF} = 1 - \left(1 - f_{n_0}\right)^{\lfloor n/n_0 \rfloor} \left(1 - f_{n_l}\right) \tag{1}$$

where $n_l = n - \lfloor n/n_0 \rfloor$.

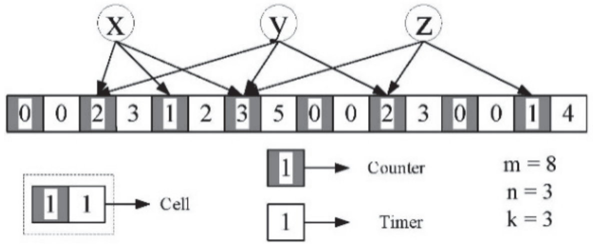


Figure 1 Data structure of TCBF component

DBFs can support the representation of dynamic ensembles, but they cannot reveal element recency which is a significant factor in OSN data. To address this issue, we propose FBF, which extends DBFs by embedding element recency information. Distinct from the DBF, we insert the time attribute into the counting Bloom filters [36]. We refer to this variant of the Bloom filter as the Time-based counting Bloom filter (TCBF), which combines two components as one cell, where each cell in the hash space consists of a counter along with a timer. Fig. 1 shows an example, a TCBF consists of *m* cells, The counter keeps the hitting times of the cell, while the timer keeps track of the last hitting time. Therefore, the counter part of FBF is the same as the DBF. Thus, we use the same parameters discussed in the DBF [16]. Therefore, the false positive rate is 0.0098, the number of cells is 1280, the bits of the counter is 4, and the number of hash functions is 7. In addition, the hash functions are uniform and independent such as SHA-1. Then, we will discuss the timer of FBF, which is distinct from the DBF. A timer reflects the insertion or update time of an element. At the design level, a representation of the timestamp at a fine granularity takes on the order of tens of bits. Distinct from many traditional web-based applications, OSN users are more interested in the recent updates of their friends [19]. We use five bits in this design to represent the approximately one-month time window. Real-world systems can use different numbers of bits to represent different time window sizes at different granularities. The FBF provides a set of user operations consisting of insertion, deletion, and membership tests, as well as the operation of the system, such as mergence. The use of FBF allows non-stop stemmed terms from a user's content to be represented in a space-efficient manner. All users maintain the FBF of each of their two-hop friends, at the first time step, each user transmits only his/her FBF to his/her neighbors, and then notify changes between the updated filter and the old filter.

3.3 FBF Ranking Model

The efficient summary of the FBF allows our keyword search scheme to handle a query in three steps. In the first step, the summary index is detected to find a set of friends that may have the keywords. In the second step, the score of friends with the most relevant responses is calculated. In the third step, severs hosting friends are ranked with the most relevant responses. In order to reduce the inter-server communication cost, our scheme can gain high query efficiency without unnecessary query throughout the entire network. Thus, the key point of our scheme is how to calculate the score of friends with relevant answers from a query. Therefore, we design a ranking model with two factors, content relevance and temporal recency.

Content Relevance, our design is based on adapting a traditional content relevent algorithm via the vector space model [37] with a variant of the TF × IDF schemes [38]. Typically, both the query and document are modeled as a vector with terms. Thus, the relevance between a query and a document can be calculated as the similarity of two vectors by using the cosine of the angle.The term frequency inverse document frequency (TF × IDF) scheme is one of the best known methods to assign term weights to a document [38]. Specifically, the TF factor is the term frequency which appeared in a document, and the IDF factor is the term frequency of the document that appeared in the corpus. To approximate the TF × IDF scheme by using the FBF in our design, we introduce a measure called inverse friend frequency (*IFF*) to replace the IDF. Therefore, to compare the TF × IDF schemes in the IR techniques, the IDF factor is the term frequency of the document that appeared in the corpus, and the *IFF* is the term frequency of the document that appeared in the FBF-based summary index. Then, *IFF* is calculated as follows:

$$IFF = \log\left(1 + \frac{|N_f|}{|N_{t,f}|}\right) \tag{2}$$

where $|N_f|$ refers to the number of friends of a user with FBF, $|N_{t,f}|$ refers to the number of term *t* that appeared in the FBF of friends. Similarly with IDF, a keyword occurring in all the friends is unnecessary for differentiating the keywords. Unlike the IDF, the *IFF* can be calculated by using a user's index locally without the need to calculate across out the entire network. With the help of the FBF, it is straightforward to calculate the parameters $|N_f|$ and $|N_{t,f}|$ conveniently.

Content Recency, as recent research by Chen and Kwak [19] shows that OSN users are more interested in recent updates from their friends, we consider the recency of paired content in the ranking model. So that membership queries can be performed against the FBF for a clause. If a query is matched, we can also get its timer, which can be computed as follows:

$$TT = \log\left(1 + \frac{T_t}{TTL}\right) \tag{3}$$

where *TT* denotes content recency in a query, T_t denotes the timer of term *t*, *TTL* denotes the max number of the timer. To give an example, we set the timer using 5 bits, so that the *TTL* of the FBF is 31, and the larger the timer, the greater the value of the timer. Then, using *IFF*, we arrive at a ranking of the relevance of content between friends of a user that can be calculated as follows:

$$R(q,\ f)=\sum_{t\in q\cap t\in FBF}\left[\log\left(1+\frac{T_t}{TTL}\right)\times\log\left(1+\frac{F}{F_t}\right)\right] \quad (4)$$

Ranking servers, due to the Key-Value-based data management, friends' data can be stored on servers on both coasts of the United States. The data of friends may be distributed in many servers across the world. In another word, a server may contain several candidate friends, so it should summarize all the candidate friends, then determine which server to contact with. Thus, ranked servers may be presented as follows:

$$R(S)=\sum_{f\in S}R(q,f) \quad (5)$$

3.4 Details of Friend Based Bloom Filter

We primarily illustrate the main operations for Friend-based Bloom filters, and illustrate how to construct the summary index for each user by leveraging FBF.

3.4.1 Operations of FBF

With an initial value of *s*, and an initial TCBF that is active. In the FBF insert operation, items are inserted in an active TCBF only and may append a new TCBF as an active TCBF when all of the ones that were previously active are full. In order to implement a FBF, we should first initialize serveral parameters: the upper bound of the FBF false probability, the largest number of TCBFs, the size of FBF, the capacity of FBF, and the number of hash functions, the bits of a counter and a timer. We note that the approaches used to initialise these parameters follow our earlier design of DBF [1].

Algorithm 1: Pseudocode for item insertion

```
Data: x is the element to be inseted into the FBF
Function: insert(x)
TTL = initial.timer()
ActiveTCBF = GetActiveTCBF()
**if** ActiveTCBF is null **then**
   ActiveTCBF = CreateTCBF(m,k)
   Add ActiveTCBF to the FBF
   s = s+1
**else**
 **for** i=1 to k **do**
   ActiveTCBF.counter[$hash_i(x)$] ++
   ActiveTCBF.timer[$hash_i(x)$] =TTL
GetActiveTCBF()
   **for** j=1 to s **do**
      **if** $TCBF_j.n_r < n_0$ **then**
         **Return** $TCBF_j$
/* Loop all hash functions k
```

The process details of the item insertion operation are shown in Algorithm 1. The FBF should start with the *TTL*, which records the base time at which the FBF is created. It then initializes the timer window to record the time of each object. If there is an item *x* to be inserted into the FBF, then the FBF must first discover an active TCBF. If no TCBFs are active, it should create a new one as an active TCBF and increments *s* by 1. If an active TCBF is found, it inserts *x* into the active TCBF, and increments the value of n_r by one for the active TCBF, let n_r be the number of elements in TCBF active. Then the counter for increments of $hash_i(x)$ by one, and the timer for the $hash_i(x)$ is set to the value *TTL*. Algorithm 1 shows the process details of the element insertion operation. The FBF must begin with the *TTL*, which records the basic time at which the FBF is generated. The timer window is then initialized to record each object's time. Thus, if there is an element *x* to be inserted into FBF, FBF first needs to discover an active TCBF. If there are no active TCBFs, it must create a new one as the active TCBF and increments by 1. When an active TCBF is found, it inserts *x* into the active TCBF, and increments the value of n_r by one in the case of the active TCBF, let n_r be the number of items in the active TCBF. The counter for increments of $hash_i(x)$ by one is then incremented, and the timer for $hash_i(x)$ is set to the *TTL* value.

Algorithm 2: Pseudocode for item query

```
Data: x is the object key for which membership is queried
Function: query(x)
timer(x) = 0
**for** i = s to 1 **do**
  counter = 0
  timer = TTL
  **for** j = 1 to k **do**
     **if** $TCBF_i$.counter[$hash_j(x)$] = = 0 **then**
        **break**
     **else**
        counter ++
        **if** (timer>$TCBF_i$.timer[$hash_j(x)$])
           timer =  $TCBF_i$.timer[$hash_j(x)$]
  **if** counter = k **then**
     timer(x) = timer
     **return** true
**return** false
/* Loop all hash functions k
```

It is convenient to represent a set *X* as FBF by repeatedly invoking the insertion operation. After the realization of the FBF, we can query with respect to the FBF instead of X. This process is sketched in Algorithm 2. If all the *counter*[$hash_i(x)$] of an item *x* are set to $1 \le i \le k$ in a TCBF, then the item *x* is a member of the *X*. Not only can we justify item *x* if in FBF, but can also obtain the temporal value of *x* by query operation if the *x* element is in the FBF. In all the *counter*[$hash_i(x)$], some may be updated by other items, so after examining all the *counter*[$hash_i(x)$], we record the lowest as the timer of item *x*, and the time value of the item *x* is time(x) = timer(x) − (current_time - *TTL*)/timer_window. Otherwise, FBF checks another TCBF, and so on. That is, *x* is a member of *X* if it does not occur in all TCBFs. The FBF supports the delete operation as well. When an *x* item is removed from the set *X*, the corresponding FBF is required to run Algorithm 3 started with *x* as an input. First, it must check to see if the *x* item is in the FBF, if there is only one TCBF that needs to be checked containing the element *x*, the value of *counter*[$hash_i(x)$] for $1 \le i \le k$ is decremented by 1. When multiple TCBFs contain the *x*, item, it is hard to distinguish which is the correct one to remove. The delete operation is denied.

```
Algorithm 3: Pseudocode for item deletion
Data: x is the object key to be deleted
Function: delete(x)
del = 0
counter = 0
for i = s to 1 do
  if TCBF[i].query(x) then
    del = i
    counter++
  if counter > 1 then   /* there are same items with different time
    break
If counter == 1 then
  for j = 1 to k do
    TCBF[del].counter[hash_j(x)] --
    if (TCBF[del].counter[hash_j(x)])
      TCBF[del].timer[hash_j(x)] = 0
  TCBF[del].n_r--
  return true
else
  return false
```

Furthermore, any two TCBFs that are active must be combined together if the sum n_r is no more than the capacity of a TCBF, which is Described in Algorithm 4. The merge operation implements in two steps: 1) Implement the addition operation between the bit vectors of the counter. 2) Set the timer to larger value.

```
Algorithm 4: Pseudocode for merge
Function: merge()
for I = 1 to s do
  if TCBF_i.n_r < n_0 then
    for k = j+1 to s do
      if TCBF_j.n_r + TCBF_k.n_r < n_0 then
        TCBF_j.counter() += TCBF_k.counter()
        if(TCBF_j.timer() < TCBF_k.timer())
          TCBF_j.timer() =TCBF_k.timer()
        TCBF_j.n_r += TCBF_k.n_r
    clear TCBF_k
```

The average time complexity of an FBF is: The addition of an element requires $O(k)$ operations, querying an item needs $O(k \times s)$ operations, member deletion needs $O(k \times s)$ operations, where k is the number of hash functions. The FBF is used as a compact summary for three important benefits as follows: 1)The FBF is a relatively efficient summarization mechanism dealing with the dynamic ensemble with no upper bound. 2)The FBF can combine together independently, so that users can trade off accuracy for storage by combining a portion of its friends' FBFs. The ability to trade accuracy independently for memory cost is especially useful for users operating on memory-constrained devices. 3)The FBF builds a new TCBF as an active TCBF only when all the active TCBF has become full, thus, we can query through the TCBFs of FBF inversely to find out which element is newly inserted compare to the old TCBFs with the timer.

3.4.2 Summary Index Dissemination Using FBF

As mentioned above, each user keeps an index table of two hop friends. This section presents how to maintain a summary index.

Privacy Issue, such as Facebook can limit access control by choosing between the four options "everyone", "friends of friends", "friends only" or "only me". Most respondents in a survey have only allowed access for friends (63%). However, there are still a large number (34%) for a user to surf in the online social network [5]. Such privacy-relevant systems [40] allow a user to control who has access to her data. Thus, instead of pulling the FBF from the summary of text documents from close friends, in our design, a user's FBF is pushed to the allowed friends.

Redundancy Issue, assuming that a user's access control is based on two hop friends, she first sends her FBF to her one hop friends, then its one-hop friends forward this FBF to all their one-hop friends except the FBF sender. As can be seen in Fig. 2, A_1 has two friends at a hop of A_2 and A_3; A_2 and A_3 are one-hop friends of each other. After A_1 has passed the message to A_2 and A_3, if none of the A_2s or A_3s know that the other is receiving the same message from A_1, they will pass the message back and forth between them. A transmission pair between A_2 and A_3 is not needed. Similarly, B_1 has two friends at a hop of B_2 and B_3, and both B_2 and B_3 have a common single-hop friend B_4, such that B_4 is B_1's two-hop friend. B_4 can receive the same message twice, one of which is from B_2, while the other is from B_3. The message transmitted over the A_2A_3 and B_3B_4 logical link is useless in these cases. Redundant messages are pure overheads, increasing network traffic. So it must avoid the transmission of redundant messages.

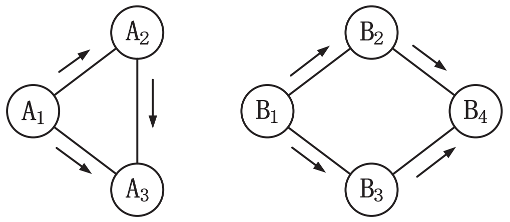


Figure 2 Unnecessary transmit on logic link A_2A_3 and B_3B_4

Summary Index Dissemination, OSNs provide a globally unique ID for each user. When a message is sent from a user, it carries the user ID and privacy control initialized with TTL hops, meaning that you allow TTL-hop friends access to your data. During a hop traversal, the message travels with its TTL value decremented by 1. That is, the message is stopped to broadcast because it is redundant or use up the TTL.

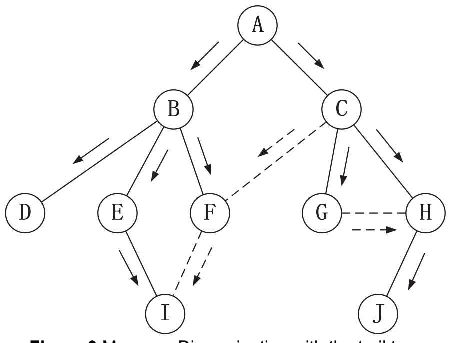


Figure 3 Message Dissemination with the trail tree

In order to avoid the redundant message, we use the path-based broadcast algorithm. We mark each link to only one hop's friends as a path. When one of the friends gets the message and it is not the first time they've received the same message, they send back a small invalidating trail link message. Thus, it will generate a spanning tree of paths covering all allowed friends rooted at the source user. Any link transmitting redundant message is excluded from the tree. The main stages of message dissemination are: 1) A small probe message traverses friends to build a spanning tree of paths. 2) The FBF message broadcasts through friends following the path tree. An example pathway tree is shown in Fig. 3. The source-user is User *A*. Solid lines are trail links, while the dashes are the redundant transmission links excluded from the trail tree. Since we only transmit the set of users' FBFs at the first time, if users update their FBFs, we will only send the changes between the updated filter and the old one [42].

4 RESULTS AND DISCUSSION

We introduce the collected Facebook trace for online social network simulation. Then, we describe the data sets used for the evaluation of the design. Finally, we describe the setup of the simulator.

4.1 Facebook Trace Collection

In particular, we adopt the BFS sampling algorithm, which discovers all users within a certain distance of the starting point. While it is likely that BFS will densely cover only a certain partial region of the graph, it can give a complete view of users' neighbors. We crawl a dataset containing 2 M users of Facebook. According to the trace analysis results, nearly 50% of the users have more than 100 friends. When extended to two hops, a Facebook user's average number of friends in two hops is 3.1×10^4, where more than 40% of users have more than 1.0×10^4 friends in two hops.

4.2 Simulator Setups

As a baseline, we develop a custom java simulator to benchmark our extraction against an algorithm with exhaustive recovery. To better represent the OSN system in the real world, we consider the tracking of real data and the characteristics of the underlying data center. To model the underlying data center, we use the fat tree network, which is a commonly used architecture in large-scale data centers [43]. As an evaluation, we tune 30000 servers using the fat tree architecture. The number of servers is initially set to 1000 and is modified to assess performance with increasing network size. Next, we randomly partition the collection of Facebook traces across these servers using Cassandra's system. Next, we randomly assigned 100 documents from WT10G [14] to each user and assigned each document a factor of time. Each user then maintains a FBF of friends within two hops. Upon the arrival of a query, we first perform a local search of the user's FBF and filter the accesses to unnessary based on the ranking algorithm outlined in section 3. To reduce the communication cost, the scheme retrieves matched documents by communicating with servers that host query anwsers.

Metrics, for the measurement of a certain query, we should consider both the system communication cost and the search accuracy of a query in a search process. Therefore, the evaluation needs to consider two types of metrics: one is the communication cost and the other is the search accuracy of results. To measure the communication cost, we use two metrics, traffic and latency. Traffic of online social network impacts heavily on the underlying network. By traffic, we mean the network resource used for text-document retrieval on servers, which is primarily a function of the consumed bandwidth of the network [15]. Specifically, during the search process, a query message is transmitted from server to server, in reality the message can traverse underlying the data centers. The traffic *TC* can be calculated as follows:

$$TC = |M| \sum_{i=1}^{N} \frac{L_i}{B_i} \tag{6}$$

where $|M|$ represents the message size of queries and answers, L_i represents the length of links and B_i represents the bandwidth for the *i*th hop in the overlay network. In this setting, traffic is the bandwidth cost of collecting all candidate results for a given query. Note that the latency of a lookup operation is the sum of the latencies of all hops in the date center network from servers to servers. Since local operations are routinely extremely fast, we ignore time spent on local operations. In order to measure search precision, we use two accepted metrics, recall and precision, which are broadly defined in keyword search. We assess the algorithm's performance by comparing its recall and precision achieved.

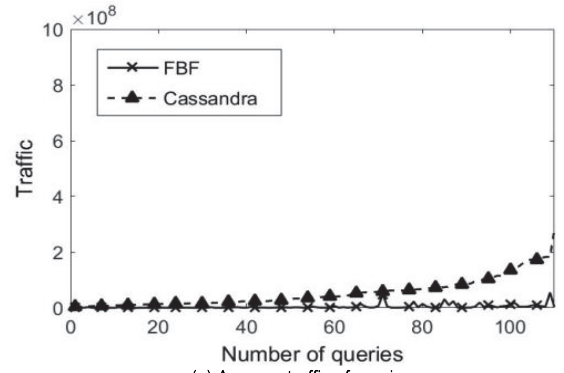
4.3 Experimental Results and Discussion

We consider the results of two aspects described in subsection 4.2, and use the FBF scheme to represent our method described in this work, while the Cassandra scheme represents the exhaustive search mentioned in section 1 that is used in the real-world social networks as a baseline for the expriments.

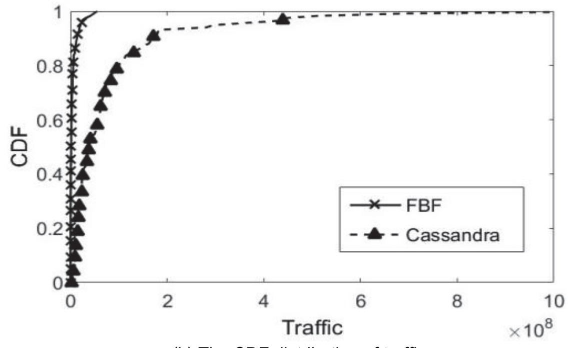
4.3.1 Results of Communication Cost

For the results, it considers first the communication cost as defined in subsection 4.2. In this subsection, the traffic and latency of a query can be seen from the experimental results.

Fig. 4a and Fig. 4b show a graphical representation of the query traffic, which shows the average traffic of queries and the CDF distribution of traffic, respectively. It can be seen that 41% of the queries using the Cassandra scheme as a baseline have a traffic count of less than 2.7×10^7, and more than 95.7% of the queries have less than 2.3×10^7 traffic by using the FBF scheme. In what follows, when we report results from our scheme named FBF for brief. Using Cassandra search, we find that the average query log traffic is 4.97×10^7, while the average amount of traffic using the FBF is only 3.8×10^6 reducing the traffic significantly by 92.4%. Therefore, we can see from the results that the communication traffic is highly reduced by using the FBF scheme compared to the Cassandra scheme.

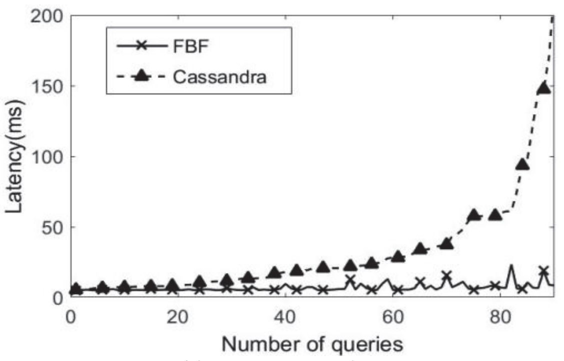


(a) Average traffic of queries

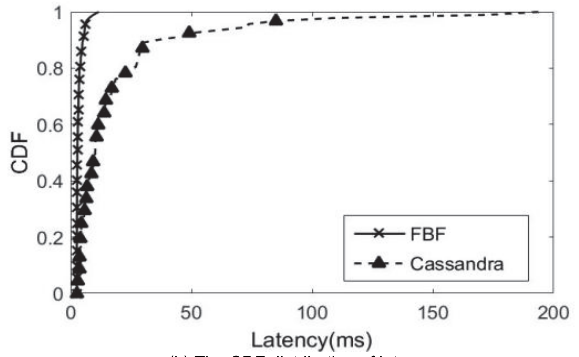


(b) The CDF distribution of traffic.

Figure 4 Results of communication traffic



(a) Average latency of queries.



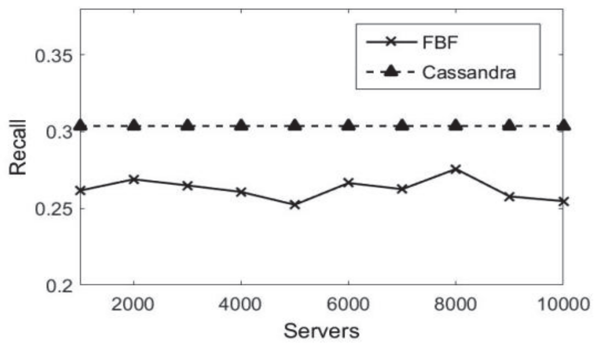
(b) The CDF distribution of latency

Figure 5 Results of communication latency

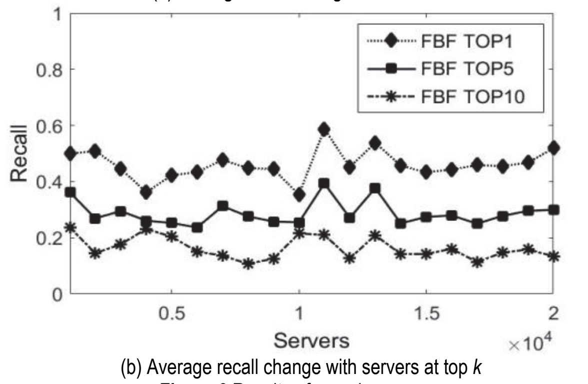
Figures show the query latency. We can see the average latency of queries from Fig. 5a and the CDF distribution of latency from Fig. 5b, where less than 60% of requests in 45.6 milliseconds by using Cassandra scheme. Using the FBF, more than 95.7% of the queries in less than 25.3 milliseconds. Thus, the average query latency using the Cassandra scheme is 64.39 milliseconds, while the mean latency using FBF is only 13.69 milliseconds resulting in a 78.7% significant latency reduction. Therefore, we can see from the results that the average latency of queries is highly reduced by using the FBF scheme compared to the Cassandra scheme. Therefore, both the communication traffic and the latency of the queries are reduced by using the FBF scheme compared to the Cassandra scheme. Then, we gain the conclusion that the communication cost of the system in a certain search process is much better by using the FBF scheme compared to the Cassandra scheme.

4.3.2 Results of Search Accuracy

It then considers the search accuracy of the results defined in subsection 4.2. In this subsection, the accuracy of the query can be seen from the experimental results. Fig. 6a plots the recall on the provided query set as the top 10 relevant documents returned using the FBF schema compare to the Cassandra. As can be seen from the figure, its performance is slightly worse than that of the Cassandra-based method, which is almost exhaustive search. We are surprised at the closeness of recall rate of requests to matching the same number of results by using the FBF scheme as compared to Cassandra, which means we can use the FBF scheme to replace the Cassandra scheme for a little reduce of recall rate. Fig. 6b shows the recall rate diagram on a set of queries provided in the returned top *k* related results. As you can see from the diagram, we set the k to 1, 5, and 10, respectively. It can be seen from the figure, that the larger *k* value is, the smaller recall rate is. It can be explained intuitively, for the larger number of results returned by a query, the more servers need be communicated in the search process. So, we usually set the value *k* to 5 in the experiments.



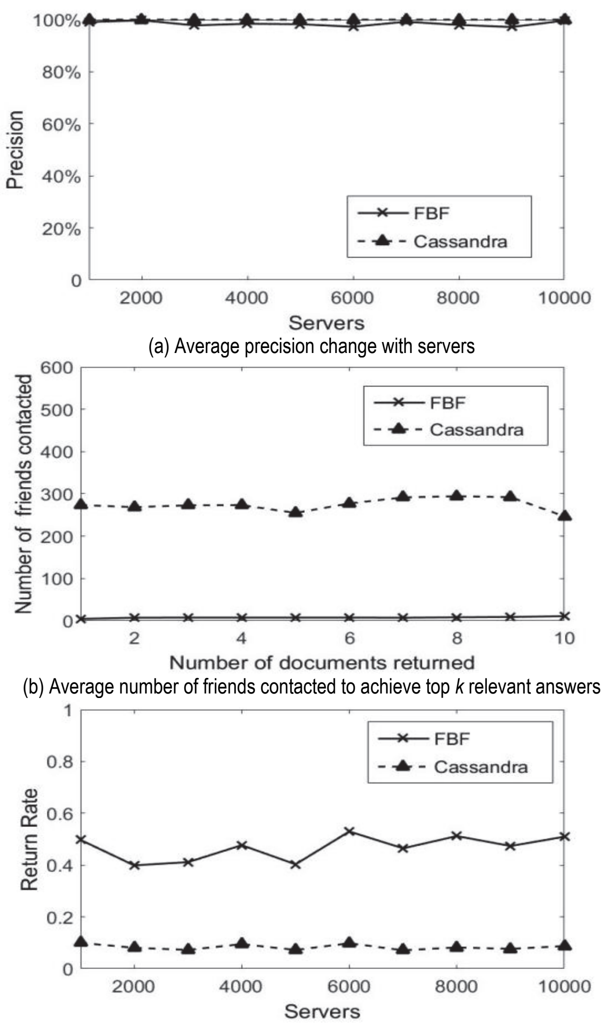
(a) Average recall change with servers



(b) Average recall change with servers at top *k*

Figure 6 Results of search accuracy

To measure a query performance, the accuracy of results is the most important metric. The accuracy of the scheme is shown in Fig. 7a. It can be seen from this figure that the average query precision is 98.3% by using the FBF scheme, while the size of the network increases from 1000 to 10000. Because of the exhaustive search for the Cassandra method, it can gain the 100% precision of the results of a query, but it need much more time for achieving this high precision. Therefore, it is a very high precision for a search system by using the FBF scheme. Otherwise, thanks to the false positive of the bloom filter mechanism mentioned in subsection 3.2, the precision of the search system by using the FBF scheme is unable to achieve 100% precision. The number of friends contacted when different numbers of top *k* results are returned is shown in Fig. 7b. From this figure, we can see that the FBF scheme achieves a mean number of 8.54 friends to contact with, whereas the Cassandra scheme needs contact with 275.7 numbers of friends. Therefore, there is a big gap between the FBF scheme and the Cassandra scheme. When we need to gain the same number of results returned for a query, the small number of friends contacted means the smaller number of servers the query system need to access. Therefore, we gain the conclusion that the FBF scheme has an incredibly low query cost compared to the Cassandra scheme in a certain query with the same number of top *k* results.



(a) Average precision change with servers

(b) Average number of friends contacted to achieve top *k* relevant answers

(c) The return rate change with servers

Figure 7 Results of search accuracy

The return rate of a query for the FBF scheme is shown in Fig. 7c, which means the rate between results returned and the number of friends contacted by a query. From this figure, it can be seen that the FBF scheme achieves an average return rate of 0.459, while the Cassandra scheme achieves 0.082. Therefore, when we need to gain the same number of top *k* results, we need to contact much smaller number of friends by using the FBF scheme compared to the Cassandra scheme. In another word, the FBF scheme has a higher search efficiency than the Cassandra scheme.

All in all, the FBF scheme has a much higher search efficiency and lower query communication cost with a little reduction of recall rate and sacrifice of query precision compared to Cassandra scheme. Therefore, the FBF scheme is a better choice for content based search systems in online social networks when it needs much lower response time with slightly lower search accuracy compared to Cassandra scheme.

5 CONCLUSION

As the online social network is a potential web-based service, more and more people are using OSNs to gain news and communication delivery along social ties between users. However, due to privacy constraints, the user is only able to access certain people in the online social network. Furthermore, because of the key-value based data management, it is inefficient to query all the relevant servers to gain the results. We have proposed a friend-based Bloom filter method with an efficient ranking model for keyword search. To evaluate our design, we perform extensive simulations using crawled Facebook traces. It is shown to achieve high efficiency process queries avoiding unnecessary server accesses, and achieves significant comunication cost reductions. This paper only discusses the application of text-based content search in social network systems. In the future, we will expand text-based content search to other kind of content such as image content. Next, we will focus on optimizing rankings models in the FBF-based summary index.

Acknowledgements

This work was partly supported by the Hubei Natural Science Foundation Innovation and Development Joint Fund Project (Nos. 2022CFD101 and 2022CFD103); in part by Xiangyang High-tech Key Science and Technology Plan Project (No. 2022ABH006848); in part by Hubei Superior and Distinctive Discipline Group of "New Energy Vehicle and Smart Transportation"; in part by the National Nature Science Foundation of China (Nos. 62362042, 62262033, and 61962029); in part by the Jiangxi Provincial Natural Science Foundation of China (Nos. 20202ACBL202005, 20224BAB202012 and 20232BAB202007); in part by the Jiangxi Provincial Social Science Foundation of China (No. 23GL52D); and in part by the Scientific and Technological Research Project of Jiangxi Provincial Education Department of China (No. GJJ2201914).

6 REFERENCES

[1] Golder, S. A., Wilkinson, D. M., & Huberman, B. A. (2007). Rhythms of social interaction: Messaging within a massive online network. *Communities and Technologies 2007: Proceedings of the Third Communities and Technologies Conference, Michigan State University 2007*, 41-66. https://doi.org/10.1007/978-1-84628-905-7_3

[2] Chen, Z., Jiang, R., & Liu, W. (2022). Nearest close friend query in road-social networks. *Computer Science and Information Systems*, *19*(3), 1283-1304. https://doi.org/10.2298/CSIS210930031C

[3] Kim, J., Yu, Y. J., & Kyung, Y. (2022). Activity-based Friend Recommendation System (ARS) Development in Location-based Social Network. *Journal of System and Management Sciences*, *12*(1), 120-128. https://doi.org/10.14704/WEB/V19I1/WEB19295

[4] Benevenuto, F., Rodrigues, T., Cha, M., & Almeida, V. (2009). Characterizing user behavior in online social networks. *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement*, 49-62. https://doi.org/10.1145/1644893.1644900

[5] Tuunainen, V. K., Pitkänen, O., & Hovi, M. (2009). Users' awareness of privacy on online social networking sites-case facebook. *Bled 2009 proceedings*, 42.

[6] Huang, J. & Jin, H. (2013). Friends Based Keyword Search over Online Social Networks. *Grid and Pervasive Computing: 8th International Conference, GPC 2013 and Colocated Workshops, Seoul, Korea, May 9-11, 2013. Proceedings 8*, 413-422. https://doi.org/10.1007/978-3-642-38027-3_44

[7] Cui, Z., Wu, Z., Zhou, C., Gao, G., Yu, J., Zhao, Z., & Wu, B. (2016). An efficient subscription index for publication matching in the cloud. *Knowledge-Based Systems*, 110, 110-120. https://doi.org/10.1016/j.knosys.2016.07.017

[8] Lakshman, A. & Malik, P. (2010). Cassandra: a decentralized structured storage system. *ACM SIGOPS operating systems review*, *44*(2), 35-40. https://doi.org/10.1145/1773912.1773922

[9] Uncu, N. (2022). Load balancing in polling systems under different policies via simulation optimization. *International Journal of Simulation Modelling (IJSIMM)*, *21*(2). https://doi.org/10.2507/IJSIMM21-2-602

[10] Bloom, B. H. (1970). Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, *13*(7), 422-426. https://doi.org/10.1145/362686.362692

[11] Baek Tejs Houen, J., Pagh, R., & Walzer, S. (2023). Simple Set Sketching. *Symposium on Simplicity in Algorithms (SOSA)*, *Society for Industrial and Applied Mathematics*, 228-241. https://doi.org/10.1145/1772690.1772735

[12] Ippolito, D., Tramèr, F., Nasr, M., Zhang, C., Jagielski, M., Lee, K., & Carlini, N. (2022). Preventing Verbatim Memorization in Language Models Gives a False Sense of Privacy. *arXiv preprint arXiv:2210.17546*. https://doi.org/10.18653/v1/2023.inlg-main.3

[13] Zhang, S., Yang, L. T., Zhang, Y., Lu, Z., & Cui, Z. (2022). Tensor-Based Forward-Backward Algorithms in Physics-Informed Coupled Hidden Markov Model. *IEEE Transactions on Artificial Intelligence*. https://doi.org/10.1109/TAI.2022.3227222

[14] Hawking, D. (2000, November). Overview of the TREC-9 Web Track. *Trec*. https://doi.org/10.1145/344250.344254

[15] Chen, H., Jin, H., Wang, J., Chen, L., Liu, Y., & Ni, L. M. (2008, April). Efficient multi-keyword search over p2p web. *Proceedings of the 17th international conference on World Wide Web*, 989-998. https://doi.org/10.1145/1367497.1367631

[16] Cheng, Y., Park, J., & Sandhu, R. (2015). An access control model for online social networks using user-to-user relationships. *IEEE transactions on dependable and secure computing*, *13*(4), 424-436. https://doi.org/10.1109/TDSC.2015.2406705

[17] Vieira, M. V., Fonseca, B. M., Damazio, R., Golgher, P. B., Reis, D. D. C., & Ribeiro-Neto, B. (2007). Efficient search ranking in social networks. *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, 563-572. https://doi.org/10.1145/1321440.1321520

[18] Mislove, A., Gummadi, K. P., & Druschel, P. (2006). Exploiting social networks for internet search. *5th workshop on hot topics in networks (hotnets06). citeseer*, 79.

[19] Chen, R., Lua, E. K., & Cai, Z. (2011, April). Bring order to online social networks. *2011 Proceedings IEEE infocom*, 541-545. https://doi.org/10.1109/INFCOM.2011.5935222

[20] Chen, Y., Yang, L. T., & Cui, Z. (2023). Tensor-Based Lyapunov Deep Neural Networks Offloading Control Strategy with Cloud-Fog-Edge Orchestration. *IEEE Transactions on Industrial Informatics*. https://doi.org/10.1109/TII.2023.3266401

[21] Busch, M., Gade, K., Larson, B., Lok, P., Luckenbill, S., & Lin, J. (2012, April). Earlybird: Real-time search at twitter. *2012 IEEE 28th international conference on data engineering*, 1360-1369. https://doi.org/10.1109/ICDE.2012.149

[22] Hu, H., Lei, W., Gao, X., & Zhang, Y. (2018). Job-shop scheduling problem based on improved cuckoo search algorithm. *Int. J. Simul. Model*, 17, 337-346. https://doi.org/10.2507/USIMM17(2)CO8

[23] Baydogan, C. & Alatas, B. (2021). Sentiment Analysis in Social Networks Using Social Spider Optimization Algorithm. *Tehnički vjesnik*, *28*(6), 1943-1951. https://doi.org/10.17559/TV-20200614172445

[24] Chen, C., Li, F., Ooi, B. C., & Wu, S. (2011). Ti: an efficient indexing mechanism for real-time search on tweets. *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, 649-660. https://doi.org/10.1145/1989323.1989391

[25] Kwak, H., Lee, C., Park, H., & Moon, S. (2010). What is Twitter, a social network or a news media?. *Proceedings of the 19th international conference on World wide web*, 591-600. https://doi.org/10.1145/1772690.1772751

[26] Termehchy, A. & Winslett, M. (2010). Keyword search over key-value stores. *Proceedings of the 19th international conference on World wide web*, 1193-1194. https://doi.org/10.1145/1772690.1772870

[27] Keith, M., Demirkan, H., & Goul, M. (2010). The influence of collaborative technology knowledge on advice network structures. *Decision Support Systems, 50*(1), 140-151. https://doi.org/10.1016/j.dss.2010.07.010

[28] Horowitz, D. & Kamvar, S. D. (2010, April). The anatomy of a large-scale social search engine. *Proceedings of the 19th international conference on World wide web*, 431-440. https://doi.org/10.1145/1772690.1772735

[29] Guo, D., Wu, J., Chen, H., Yuan, Y., & Luo, X. (2009). The dynamic bloom filters. *IEEE Transactions on Knowledge and Data Engineering*, *22*(1), 120-133. https://doi.org/10.1109/TKDE.2009.57

[30] Kwak, H., Lee, C., Park, H., & Moon, S. (2010, April). What is Twitter, a social network or a news media?. *Proceedings of the 19th international conference on World wide web*, 591-600. https://doi.org/10.1145/1772690.1772751

[31] Zhang, R. H., Osma, M. N., Yaakup, H. S. B., & de Costa, F. (2023). A social media based study on the motivation of fans' production behavior. *Journal of Logistics, Informatics and Service Science, 10*(1), 269-279. https://doi.org/10.33168/JLISS.2023.0115

[32] Nguyen, H. H., Imine, A., & Rusinowitch, M. (2016). Private Link Exchange over Social Graphs. *arXiv preprint arXiv:1609.01616*.

[33] Niu, B., Zhang, T., Zhu, X., Li, H., & Lu, Z. (2014). Priority-aware private matching schemes for proximity-based mobile social networks. *arXiv preprint arXiv:1401.8064*.

[34] Beierle, F. (2018). Do you like what I like? Similarity estimation in proximity-based mobile social networks. *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, 1040-1047. https://doi.org/10.1109/TrustCom/BigDataSE.2018.00146

[35] Im, N. & Bang, H. (2022). An empirical study on the antecedents of knowledge hiding behavior. *Journal of Logistics, Informatics and Service Science*, *9*(4), 209-222.

[36] Fan, L., Cao, P., Almeida, J., & Broder, A. Z. (2000). Summary cache: a scalable wide-area web cache sharing protocol. *IEEE/ACM transactions on networking*, *8*(3), 281-293. https://doi.org/10.1109/90.851975

[37] Salton, G., Wong, A., & Yang, C. S. (1975). A vector space model for automatic indexing. *Communications of the ACM*, *18*(11), 613-620. https://doi.org/10.1145/361219.361220

[38] Salton, G., Fox, E. A., & Wu, H. (1983). Extended boolean information retrieval. *Communications of the ACM*, *26*(11), 1022-1036. https://doi.org/10.1145/182.358466

[39] Kalogiannidis, S., Chatzitheodoridis, F., Savvidou, S., Kagioglou, F., & Macedonia, W. (2022). The Impact of Online Communications on Different Users' Social, Emotional, and Moral Competence as a Potential Business Communication Tool. *Journal of System and Management Sciences*, *12*(5), 359-373. https://doi.org/10.33168/JSMS.2022.0521

[40] Jeong, S. J. & Kim, B. M. (2021). Network analysis of social awareness of media education for primary school students studied through big data. *Computer Science and Information Systems*, *18*(2), 575-595. https://doi.org/10.2298/CSIS200316011J

[41] Cui, Z., Lu, Z., Yang, L. T., Yu, J., Chi, L., Xiao, Y., & Zhang, S. (2023). Privacy and Accuracy for Cloud-Fog-Edge Collaborative Driver-Vehicle-Road Relation Graphs. *IEEE Transactions on Intelligent Transportation Systems*, *24*(8), 8749-8761. https://doi.org/10.1109/TITS.2023.3254370

[42] Eppstein, D., Goodrich, M. T., Uyeda, F., & Varghese, G. (2011). What's the difference? Efficient set reconciliation without prior context. *ACM SIGCOMM Computer Communication Review*, *41*(4), 218-229. https://doi.org/10.1145/2018436.2018462

[43] Niranjan Mysore, R., Pamboris, A., Farrington, N., Huang, N., Miri, P., Radhakrishnan, S., & Vahdat, A. (2009, August). Portland: a scalable fault-tolerant layer 2 data center network fabric. *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, 39-50. https://doi.org/10.1145/1592568.1592575

Contact information:

Jinzhou HUANG, Assoc. Prof., PhD
School of Computer Engineering, Hubei University of Arts and Science,
No. 296, Longzhong Road, Xiangyang, 441053, China
E-mail: huangjinzhou@hbuas.edu.cn

Yan TONG, Assoc. Prof., PhD
College of Science, Huazhong Agricultural University,
No. 1 Shizishan Street, Hongshan Distr, Wuhan, 430070, China
E-mail: tongyan.cherish@hotmail.com

Bo HANG, Prof., PhD
School of Computer Engineering, Hubei University of Arts and Science,
No. 296, Longzhong Road, Xiangyang, 441053, China
E-mail: bohang@hbuas.edu.cn

Degang XU, Assoc. Prof., PhD
School of Computer Engineering, Hubei University of Arts and Science,
No. 296, Longzhong Road, Xiangyang, 441053, China
E-mail: pcxinx@163.com

Feng WANG, Assoc. Prof., PhD
School of Computer Engineering, Hubei University of Arts and Science,
No. 296, Longzhong Road, Xiangyang, 441053, China
E-mail: wangfeng@hbuas.edu.cn

Jing YU, PhD
(Corresponding author)
School of Computer and Big Data Science, Jiujiang University,
No. 551, Qianjin East Road, Jiujiang, Jiangxi 332005, China
E-mail: yujingellemma@gmail.com