IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 21, Issue 2, Ser. II (Mar - Apr 2019), PP 23-27
www.iosrjournals.org

Construction and Compression of the Social Inverted Index for Multiple Social Tagging Systems

Miss Vahida D. Havaldar[1], Prof B.D.Jitkar[2]
1D.Y. Patil College of Engg and Technology, Kolhapur, Maharashtra
2Prof. D. Y. Patil College of Engg. and Technology, Kolhapur, Maharashtra
Corresponding Author: Miss Vahida D. Havaldar

Abstract: *The use of inverted index and compression techniques is partially accountable for the current performance achievement of web search engines. In this paper, we discuss how to crawl tagging information from multiple social tagging systems and to store it according to classes of properties of UTO. Also, we introduce an indexing technique for tagging information of social tagging systems and compression technique to reduce its size. We evaluate the performance against three factors – time required to build index, space required to store index, time required to process the query.*

Keywords: *index, tag , social tagging, compression, crawler*

Date of Submission: 27-02-2019 Date of acceptance: 13-03-2019

I. Introduction

The Social tagging systems are a family of web 2.0 applications where users can upload, share resources (e.g. videos, music, photos etc) with other users and annotate them with a list of freely chosen keywords called tags. Social tagging system (STS for short) promotes decentralization of content control and leads the web to be more open and democratic environment. These systems are widespread with millions of people using them daily to organize and retrieve online content. STS such as Delicious, BibSonomy, Last.fm, Flicker etc., bring people together through their shared interests, e.g., URLs of websites in Delicious, BiBETeX entries in BibSonomy, photos in Flicker, sound tracks in Last.Fm etc. The primary goal of tags is to help individual users to organize and retrieve their own content. The major STSs such as BibSonomy rely on RDBMS for query processing. The inverted index is a data structure used for query processing in keyword–based web search, which provides access to the list of documents that contain the given term. It is crucial to tag–based web search but it presents an obstacle to its use in the social−tagging system. In tag–based web search the user serves as an additional dimension, namely user dimension. A resource is annotated with tags by user, creating ternary relationship among resource, tag and user that cannot be entirely contained in a normal inverted index. In order to preserve this ternary relationship new type of inverted index, namely social inverted index has been designed [2].

II. Motivation

The Social inverted index fully supports the social dimension of social tagging by adding user sub list to each resource in resource posting list. It highlights the value of user who participated in tagging and preserves ternary relationship among user, resource and tag. A user's tagging activity reflects the user's own preferences and interests. This helps in improving information retrieval tasks as tags act as meaningful keywords bridging the gap between humans and machines.

III. Literature Review

A. Using annotations in enterprise search

Dmitriev et al. [5] built an inverted index that handles the terms separately from the contents, anchor texts, and tags of each page. However, this approach does not consider incorporating individual user-tagging information in the inverted index.

B. Efficient network aware search in collaborative tagging sites.

Yahia et al. [6] manage one inverted list per (tag, seeker) pair to facilitate easy and fast access to the seeker-dependent scores that are assigned to each item. Even though this approach takes into account the user aspect of searching in the inverted index, it does not consider incorporating individual user information into the inverted index.

C. ***Hexastore: sixtuple indexing for semantic web data management.***

A representative approach is Hexastore [7], which is an indexing scheme for managing a large amount of RDF triples by indexing them in six possible ways, one for each possible ordering of the three RDF elements s, p and o. Because Hexastore is designed to guarantee the scalability of RDF query processing, it cannot be exploited in search query processing.

*D.**A social inverted index for social tagging-based information retrieval***

Kang-Pyo Lee [2] build basic social inverted index with a tag index and the corresponding resource posting lists and user sublists for social tagging IR. But it considered only naïve frequency weights.

IV. Challenges Faced By Current Social Tagging Systems

A. The space cost associated with user sub lists and resource posting list. More efficient and compact index representations are needed to enhance the performance of indexing.
B. A new index structure that integrates more than one STS is needed. Currently, the social inverted index can handle only one social tagging system. In order to extend the coverage of tag based search engines multiple social tagging systems need to be integrated.
C. The current implementation of social inverted index uses web crawling or open API to collect tag data. A more intelligent crawling algorithm that can automatically detect, collect and index tag data spread on the social web should be devised.

V. Proposed Solution

Since we want to integrate multiple Social tagging systems (STS), the designed crawler will be targeted over two or more tagging systems to detect and collect tag data as shown in figure 1. Once the tagging data has been extracted, it will be given as input to index construction module. If crawling is performed after the index is constructed then existing index will be updated using merge based methodology. After index has been constructed, it will be compressed to reduce extra space cost.

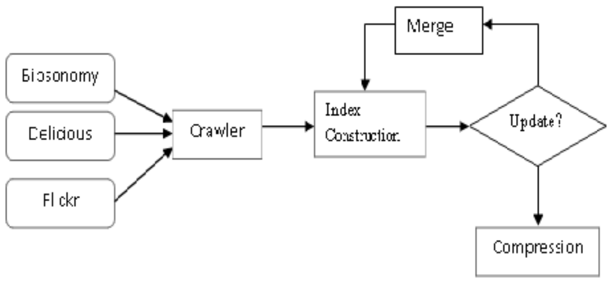

Fig. 1: Architecture of proposed system.

A. Social Tagging System Crawler

To integrate multiple STS (Delicious, Flicker, BibSonomy etc.), a tag crawler based on Upper Tag Ontology (UTO) [9] will be used. The crawler will extract and store tag data from multiple STS in RDF (Resource Description Framework)[10] triplet format according to Upper tag ontology. The crawler has six major components. The main components act as host to initiate other components. The model component is responsible for link logic that records links already visited. The filter component evaluates links and indicates which should be visited and which should be ignored. The crawler component coordinates executed tasks and distributes them to multiple threads. The parser component extracts the tagging data from HTML codes. And, finally, the RDF store stores extracted tagging data according to the classes and properties of the UTO.

A.1 Crawling Technique for BibSonomy:

I. Access the tag cloud at http://www.bibsonomy.org/tags.
II. Visit every tag in the tag cloud.
III. For each TagA in the tag cloud
a. Visit <http://www.bibsonomy.org/tags/tagA>.
b. Parse the HTML code to grab information about bookmarks, taggers and related tags.
c. For each bookmark the crawler will go to <http:// http://www.bibsonomy.org/url/idOfURL> and crawl the history of the bookmarks, focusing on which users had tagged this bookmark on which date(s).
d. After gathering data about all of the bookmarks on the first page for TagA, the crawler will visit the second and subsequent pages for TagA.

ii. The crawler will then repeat this process for TagB and all subsequent tags in the cloud.

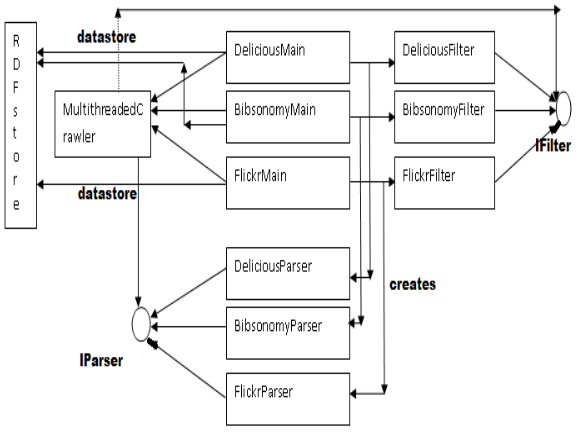

Fig. 2: STS crawler

A.2 Crawling Technique for Delicious:

i. Start with the cloud at <http://delicious.com/tags>.
ii. Visit every tag in the tag cloud.
iii. For TagA in the tag cloud,
a. Visit <http://delicious.com/tags/tagA>.
b. Parse the HTML code to grab information about bookmarks, taggers and related tags.
c. For each bookmark the crawler will go to <http://delicious.com/url/idOfURL> and crawl the history of the bookmarks, focusing on which users had tagged this bookmark on which date(s).
d. After gathering data about all of the bookmarks on the first page for TagA, the crawler will visit the second and subsequent pages for TagA.
iv. The crawler will then repeat this process for TagB and all subsequent tags in the cloud.

A.3 Crawling Technique for Flickr:

i. Access the tag cloud at <http://flickr.com/photos/tags> and
ii. Visit each tag in the cloud.
iii. For each tag page <http://flickr.com/photos/tags/tagA>
a. Collect information about related tags.
b. Visit each image on the tag page.
c. Collect information about the image, the tags and the tagger.
iv. The crawling process will continue with <http://flickr.com/photos/tags/tagA/? page=2>.
v. To avoid duplicate visits, only links of the form <http://www.flickr.com/photos/taggerID/PhotoID/> will be accepted.

B. Construction of tag to resource Social inverted index

To construct social inverted index Hadoop MapReduce framework will be used. The input to the index construction engine is in rdf format generated by STS crawler. It is split into multiple parts and each part is fed to a mapper. Individual parts are processed in parallel. Each map task involves assigning an ordinal number to each resource as a resource identifier and mapping resource IDs to their corresponding URLs or URIs. We will use tag names and user IDs to identify each tag and user respectively. After mapping has been done, the mapper iterates all over the tags and the weight values are calculated.

In shuffle and sort phase, the execution framework brings together all resource postings and user IDs that belong to same tag and sorts each list in increasing order. This simplifies job of reducer which simply needs to gather all information without any redundancy and write on to disk.

As shown in figure 4, there are three major components in the social inverted index a tag index, resource posting list and user sublist. Each component will be stored in separate file. A tag index is a set of index tags in lexicographical order that consists of three subcomponents a tag name, the weight of tag (w_t, R) and pointer to the head of corresponding resource posting list. Each resource posting list represents resources that are annotated with a tag and a node in the list has three subcomponents: a resource identifier, the weight of the resource annotated with the tag ($w_{t, r}$) and a pointer to the head of the corresponding user sublists. Each user sublist represents those users who annotate a resource with a tag and a node in the sublist has two subcomponents: a user name, weight of the tag ($w_{t, r, u}$) annotating the resource with the tag.

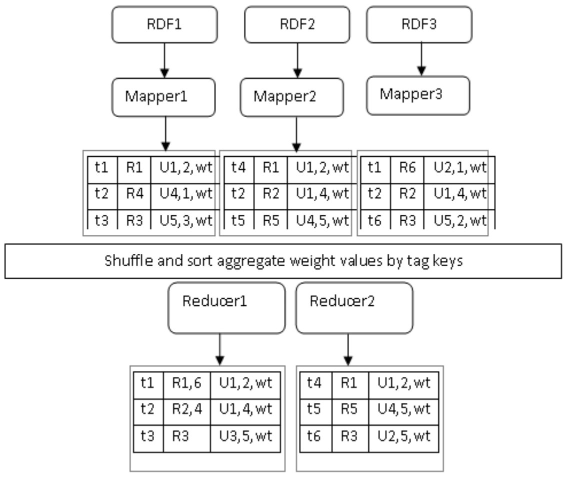

Fig. 3 : Illustration of Social Inverted index construction process in Hadoop MapReduce Framework with three mappers and two reducers.

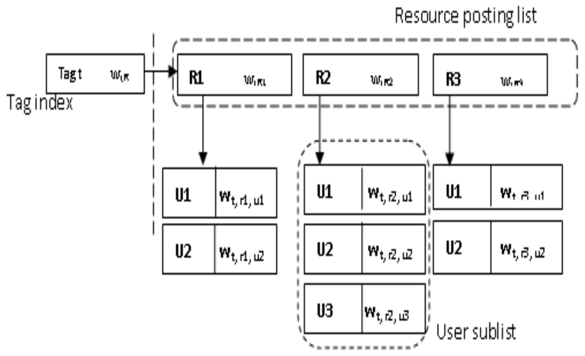

Fig. 4: A part of social inverted index

C. Index Compression

Given a block of n integers, FOR partitions it into m distinct frames. A frame is a range of possible values in block, typically [0, max], where max is the largest value in group of delta values. FOR needs $\log_2$ (max+1) bits, called as bit frame, to encode each integer in a block. The compression and decompression is done using highly optimized routines. Each routine is loop- unrolled to encode or decode m values using shift and mask operations only. The selection of the appropriate routine is done using a precomputed lookup table. The bit frame is stored using one byte in the block header and the compression routine is executed to encode the block. During decompression, FOR reads the bit frame, performs one table lookup to select the decompression routine and executes iteratively the routine over the compressed block.

D. Index maintenance and query processing

Each time crawling is performed and social inverted index is constructed, the existing index will be updated using new set of triplets. The steps are:

i. Identify the new triplets that have been added since the last update using the timestamp information of tagging.
ii. Build a social inverted index with these new triplets.
iii. Merge this (small) social inverted index with the existing social inverted index.

To support union and intersection search operations on inverted list, search algorithms will be implemented [1]. Two measures reflecting the social semantics of tagging are tag frequency and tag co−occurrence frequency. As a social inverted index is being built, tag frequencies of any kind can be computed in a straight–forward way.

Tag frequency of tag t in r ,that is, ($f_{t,\,r}$)is the number of users who annotate r with t.($U_{t,r}$)

$$f_{t,\,r} = \left| U_{t,r} \right| \qquad (1)$$

The resource frequency of t in the collection is the number of resources that are annotated with t.(R_t)

$f_{t, R}= | R_t |$ (2)

The total frequency in the collection is sum of the frequencies of t in all resources.

$F_{t,R}=\sum f_{t, r}$, for all $r \in R_t$ (3)

where $f_{t, r}$ is tag frequency see Eq. (1)

The tag co−occurrences can be viewed at two levels: the macro level and the micro level. Two tags form a macro level co–occurrence if they are assigned to same resource and they form a micro level co−occurrence if they are assigned to same resource by same user. The social inverted index makes it easy to calculate the tag co–occurrence frequency at both levels

Macrocof(t1 , t2)= $| R_{t1} \cap R_{t2} |$, for all $r \in R_{t1} \cap R_{t2}$

Microcof(t1 , t2)= $\sum | U_{t1,r} \cap U_{t2,r} |$, for all $r \in R_{t1} \cap R_{t2}$

VI. Conclusion

In this paper, we addressed that the social tagging systems need a new compact indexing structure which integrates multiple social tagging systems and uses an intelligent crawling algorithm that can automatically detect and collect tagging data spread over social web should be devised. Further, we discussed crawling algorithms for three different social tagging systems namely, Delicious ,BibSonomy and Flickr. The crawler extracts and integrates tagging data and stores in RDF format according to classes and properties of UTO. Once tagging data has been collected an index can be constructed and compressed to reduce its extra space cost. Finally, we discussed index maintenance and query processing.

Refrences

[1]. Hao Wu, Guoliang Li, and Lizhu Zhou. Ginix: Generalized Inverted Index for Keyword Search. IEEE TRANSACTIONS ON KNOWLEDGE AND DATA MINING VOL: 8 NO: 1 YEAR 2013.

[2]. Kang-Pyo Lee, Hong-Gee Kim, Hyoung-Joo Kim. A social inverted index for social tagging-based information retrieval.Journal of Information Science 2012.

[3]. Gonzalo Navarro , Edleno Silva de Moura, Marden. Adding Compression to Block Addressing. IEEE TRANSACTIONS 2000.

[4]. Renaud Delbru, Stephane Campinas, Krystian Samp, Giovanni Tummarello. ADAPTIVE FRAME OF REFERENCE FOR COMPRESSING INVERTED LISTS, IEEE TRANSACTIONS 2010.

[5]. Dmitriev PA, Eiron N, Fontoura M, Shekita E. Using annotations in enterprise search. In: Proceedings of the 15th International World Wide Web Conference. Edinburgh, UK, 2006, pp. 811–817.

[6]. Yahia SA, Benedikt M, Lakshmanan LVS, Stoyanovich J. Efficient network aware search in collaborative tagging sites. Proceedings of the VLDB Endowment 2008; 1(1): 710–721.

[7]. Weiss C, Karras P, Bernstein A. Hexastore: sextuple indexing for semantic web data management. Proceedings of the VLDB Endowment 2008; 1(1): 1008–1019.

[8]. Matteo Catena, Craig Macdonald, Iadh Ounis. On Inverted Index Compression for Search Engine Efficiency M. de Rijke et al. (Eds.): ECIR 2014, LNCS 8416, pp. 359–371, 2014. Springer International Publishing Switzerland 2014.

[9]. Ying Ding, Elin K. Jacob, Michael Fried, Ioan Toma, Erjia Yan, Schubert Foo. Upper Tag Ontology (UTO) For Integrating Social Tagging Data. Journal of the American Society for Information Science and Technology, Volume 61 Issue 3, March 2010, Pages 505-521.

[10]. Graham Klyne, Jeremy J. Carroll. Resource Description Framework(RDF): Concepts And Abstract Syntax. IEEE TRANSACTIONS 2006.

IOSR Journal of Computer Engineering (IOSR-JCE) is UGC approved Journal with Sl. No. 5019, Journal no. 49102.

Miss Vahida D. Havaldar. " Construction and Compression of The Social Inverted Index for Multiple Social Tagging Systems." IOSR Journal of Computer Engineering (IOSR-JCE) 21.2 (2019): 23-27.