Collaborative Data Publishing and Searching System

Beng Chin OOI[1], Bei YU[1], and Guoliang LI[2]
[1] National University of Singapore
[2] Tsinghua University

***Abstract*— In this paper, we present a folksonomy-based collaborative data publishing and searching system. The system accepts data objects described with user-created metadata, called *data units*. The system supports flexible structure on the data units, and places no restrictions on the vocabulary used. We devise a generic table model for storing and representing the data units of various structures. We propose a framework for managing the data units and providing browsing, searching and querying services over them. We present our current approaches and discuss relevant research issues.**

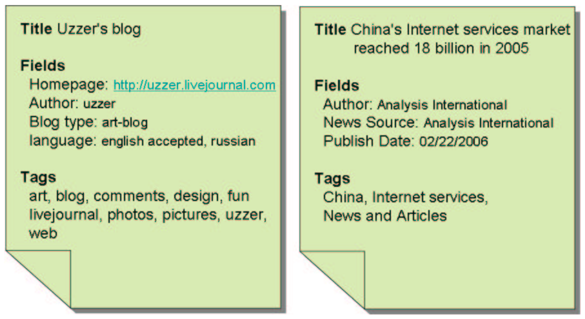
***Index Terms*— folksonomy, tagging, clustering.**

I. Introduction

Digital information publishing and searching becomes increasingly necessary in recent years, due to the popularity of the Internet services. We have witnessed the growth of a number of such web services including Google Base [3], Delicious [1], liveplasma [4], and flickr [2]. While these systems vary in their particular services provided, they share the same operation mode — users publish data items such as URLs, pictures, advertisements, etc. that are associated with simple descriptions such as tags and attributes created by them. The system organizes the published data items based on users' own descriptions and makes them accessible. For example, Delicious allows an user to bookmark URLs in its server, and requires each URL to have a set of user-provided tags (a.k.a. keywords). In some sense, the users collaboratively contribute tags to the system, which are then used to categorize the URLs to facilitate webpage searching (by browsing or querying the tags). Such collaborative but unsophisticated way of organizing information with user-created metadata is coined as *folksonomy* (combination of "folk" and "taxonomy"), and such systems are sometimes called the collaborative tagging systems [9]. A distinguishing feature of these systems is that the user plays the dominant role — the collaborative behavior of the users decides the data semantics and organization of the systems. Although the lack of controlled vocabulary and systematic taxonomy of the concepts makes the classification of the data objects in these systems imprecise and imperfect, they are convenient to users and easy to deploy, and more importantly, they can adapt to the dynamic changes of the Web content.

Besides simple tags, richer and flexible data structure could be used to describe a published item to provide more powerful expressiveness to the user. For example, Google Base allows users to define their own attributes, and to describe their published objects with variable number of attributes. The freedom from a strict syntax of the published data items is very convenient to users. However, it is a challenging task to organize and classify the data items with variable "schemas" and topics, in order to make them searchable. In Google Base, a list of types such as *products*, *recipes*, are provided, and users are encouraged but not restricted to publish their data item to a specific type. The way Google Base stores and organizes the data items is unknown, and it takes 15 to 60 minutes to make a newly published data items searchable in its website.

In this paper, we describe our attempt to build a general system framework for supporting collaborative publishing and searching services. The system accepts data objects described with user-created metadata, stores and classifies them, and provides various querying and searching interfaces such as browsing, keyword search, and structured query. The metadata created by users can be both for their own use (for labeling their published information to the system) and for the system to use to organize all the published information. The data unit, that users use to describe their published information of any kind, consists of *title*, a number of *fields*, and a set of *tags*. Basically, a *field* is an attribute/value pair for characterizing certain property of an object, e.g., *color:yellow* for a puppy. A *tag* is a word or a phrase the user uses as "keyword" to characterize the published object. For example, the tags of a puppy could be *animal*, *dog*, etc.. [9] discusses 7 types of tags an user uses to label URL bookmarks on Delicious website. Figure 1 shows two example data units that describe different types of information. To illustrate, the left data item describes the blog of uzzer. It uses four fields for showing the location, author, type, and language of the blog. In addition, it has 9 tags that are "keywords" of the blog.


Title Uzzer's blog

Fields
Homepage: http://uzzer.livejournal.com
Author: uzzer
Blog type: art-blog
language: english accepted, russian

Tags
art, blog, comments, design, fun
livejournal, photos, pictures, uzzer,
web

Title China's Internet services market reached 18 billion in 2005

Fields
Author: Analysis International
News Source: Analysis International
Publish Date: 02/22/2006

Tags
China, Internet services,
News and Articles

Fig. 1. Examples of data units.

Within our system framework, we propose a data model for storing and representing the collection of data units accepted from users. It includes a single generic table for storing the data units, and a set of virtual relations as views of the generic table for representing different topics of the data stored in the generic table. The user browses and queries over the virtual views, and the system retrieves results from the generic table. Our generic table model differs from the universal relation model [10], [8] proposed and studied earlier in two main points. First, the universal relation is designed for logical representation of an application domain in order to free the user from dealing with specific access paths when issuing queries, while our generic table is the schema for physically representation and store of the tuples, which is not visible to the user. Second, compared with that the universal relation schema describes a specific application domain, our generic table model is for comprehensively storing data of all types of domains.

Our proposed system framework also includes a data units categorizer that dynamically clusters and assigns incoming data units into various virtual relations according to their different topics and structures, a multi-function query processor for dealing with various kinds of queries, a storage manager for storing and indexing the data units whose volume may grow quickly depending on the popularity of the system. Figure 2 illustrates the architecture of our system.

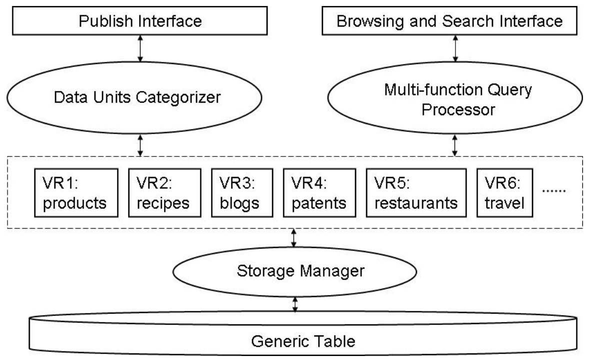


Fig. 2. System architecture. (VR is the short for Virtual Relation.)

II. Data Model

The data model in our system is a generic relational table and a set of virtual relations that are views over the generic table. Data units published to the system are all physically stored in the generic table. Different from traditional relational database design, the schema of the generic table in our system is designed in an ad-hoc and dynamic way. It contains a set of fixed attributes and a set of non-fixed attributes. Fixed attributes are defined by the system, while non-fixed attributes are dynamically inserted according to the data units published to the system. That is, its schema is collaboratively decided by users. The single table model is suitable for storing the data units in our system because there is no predictable data dependencies among attributes collaboratively defined by mass users.

Definition 1 *The generic relational table schema is an expression of the form $R(U)$, where R is the name of the table, and U is the set of attributes such that $U = U_F \cup U_N$ and $U_F \cap U_N = \phi$. $U_F = \{A_1, A_2, \cdots, A_m\}$ is the finite set of fixed attributes, $U_N = \{A_{m+1}, A_{m+2}, \cdots\}$ is the infinite set of non-fixed attributes.*

The domains of the attributes in U_N are initially undefined, and assumed to be infinite and stored as string format. When the generic table is populated with enough number of tuples, we can use machine learning approach to learn the patterns and statistics of the values in order to determine the domains of attributes.

In our system, the fixed attributes set U_F contains: id, $author$, $title$, and $tags$, where id is system created when a new data unit is inserted and it is the primary key of the table, $author$ and $title$ correspond to the person who publishes the data unit and the title of it, and $tags$ is a textual attribute for storing all the tags listed in the data unit. When the system initializes, non-fixed attributes set U_N is empty. More and more attributes will be automatically added to it when data units having new attributes are published to the system continuously. On the other hand, when obsolete data units are deleted from the system, some attributes may also be removed from U_N.

The generic table is the schema for storing data units in our system, but it cannot be semantically meaningful to users, because the data units stored in it are very diverse in their topics. In our system, the user poses queries in terms of a set of virtual relations, which are defined as views of the generic table. A virtual relation schema is mapped to U_F and a subset of the attributes in U_N of the generic table. The set of virtual relations $(R_1, R_2, \cdots, R_n)$ defined in the system is called the *virtual schema* (or *semantic schema*) of the generic table.

Definition 2 *A virtual relation $R'(U)$ of the generic table $R(U_F, U_N)$ is defined as a view with a query q_I over instance I of $R(U_F, U_N)$, denoted as $R'(U) \mapsto R(U_F, U_N) = q_I$.*

In the generic table, $null$ values are allowed. The semantics of the $null$ is treated as *inapplicable*, and the operations on $null$ values are the same as in traditional relational model. In particular, if a tuple has $null$ values in all the columns, it is called *null* tuple. If the instance I of a relation contains all *null* tuples, I is regarded as $null$, i.e., $I = null$.

Definition 3 *Given two (virtual) relations $R(U)$ and $R'(U')$, and supposing $V = U \cap U'$, two tuples $u \in R(U)$ and $u' \in R'(U')$ are equivalent if $u(V) = u'(V)$, $u(U - V) = null$ and $u'(U' - V) = null$, denoted as $u \equiv u'$.*

When the system adds an attribute to U_N of the generic table, all tuples of the current instance I are assigned $null$ values for the new attribute, and they maintain equivalence with their original forms.

Definition 4 *Given an instance I of $R(U_F, U_N)$, an attribute $A \in U_N$ is unnecessary if $\pi_A(I) = \{t[A] | t \in I\} = \{null\}$.*

Unnecessary attributes may appear in generic table or virtual relations when obsolete tuples are deleted, and they need be automatically removed.

The basic operations on the generic table and virtual relations include selection (σ), projection (π), join ($\bowtie$), union ($\cup$), and difference ($-$), which are the same as in traditional relation model. Based on the basic operations, we define an augmented selection operator called *reducible selection*, denoted as σ^r. Given a relation $R(U)$ and its instance I, and supposing $A \in U$ and $a \in dom(A)$, σ^r takes as input relation I and returns the relation: $\sigma^r_{A=a}(I) = \pi_V(\sigma_{A=a}(I))$, where $V \subseteq U$ and $U - V$ is the set of all unnecessary attributes of $\sigma_{A=a}(I)$. That is, the resultant relation of reducible selection will have no unnecessary attributes.

Theorem 1 *Supposing c is a selection condition on a subset of columns of relation $R(U)$ whose instance is I, the two selection operations $\sigma_c(I)$ and $\sigma^r_c(I)$ produce two relations consisting of equivalent tuples.*

In addition to ordinary comparison operators such as $=$, $<$, $>$, there are also textual-based comparisons in our data model, since most data units have lots of textual fields, e.g., *tags* attribute. Given a set of keywords $K = (k_1, k_2, \cdots, k_q)$, a relation $R(U)$ and its instance I, and a subset of textual attributes $V \subseteq U$, for each tuple $t \in I$, the textual comparison operator $match(t[V], K)$ returns a score $score(t[V], K) \in [0, 1]$ indicating its relevance to the keywords. The $match$ operator relies on a fulltext index on the keywords of the values of textual attributes. Given a threshold τ, $\sigma_{match(V,K)>\tau}(I)$ will return all tuples in I whose scores are matched higher than τ.

The query Q over the generic table and virtual relations is constructed by combining the basic operators.

Definition 5 *A virtual schema $\{R_1(U_1), R_2(U_2), \cdots, R_n(U_n)\}$ is complete in terms of an instance I of the generic table $R(U_F, U_N)$ if $I(R_1(U_1)) \bowtie_o I(R_2(U_2)) \bowtie_o \cdots \bowtie_o I(R_n(U_n)) = I$, where $\bowtie_o$ denotes outer join, and $I(R_i(U_i)) (1 \le i \le n)$ is the instance of $R_i(U_i)$ generated by evaluating query q_i, which is the view defined for $R_i(U_i)$, over I.*

With complete virtual schema, the content stored in the generic table can be fully exposed to users.

III. System framework

As a collaborative publishing and searching system, our system initializes with an empty generic table *AllUnits* with system created attributes, defined as

$$AllUnits(id, author, title, tags).$$

As users publish data units to the system, the generic table is populated with more and more tuples, and consequently it will have more and more attributes.

When storing a data unit into the generic table, we represent it as a tuple according to the schema of the generic table. If there are attributes defined in the data unit that are not in $U_N \in AllUnits$, the system will add those attributes to U_N during the insertion of the new tuple. For example, if upon the system initialization, the two data units shown in Figure 1 are published to the system one at a time, the resultant generic table schema is *AllUnits(id, title, author, tags, **homepage, blog type, language**)* after the first data unit is inserted, and it becomes *AllUnits(id, title, author, tags, homepage, blog type, language, **news source, publish date**)* after the second one is inserted. A data unit becomes a tuple in the generic table after it is stored in the system. In the rest of the paper, we will use data unit and tuple interchangeably for referring to the tuples stored in the tables.

The generic table is maintained by the storage manager component (Figure 2), and it is not visible to the user. Users access the stored data through the virtual relations over the generic table, which are built and updated incrementally as new data units are published to the system. Each virtual relation should represent a semantic category meaningful to the user. For example, a virtual relation $blog(blog\_name, blog\_type, homepage)$ represents a category of data units describing blogs. Since the domains of published data units is unrestricted, constructing the virtual relations is identified as the task of incrementally clustering the incoming data units into various virtual relations. This task is performed by the data unit categorizer depicted in Figure 2.

Therefore, our system need perform several actions when accepting a new data unit. First, the data unit classifier either assigns the new data unit to an existing virtual relation, or creates a new virtual relation with the new data unit as the only tuple. Then, the data unit is passed to the storage manager, which actually inserts the new data unit into $AllUnits$, and updates the mapping between the virtual relation accepting the new tuple and $AllUnits$ if the new tuple causes changes of the schema of $AllUnits$.

We intend to provide in the multi-function query processor (Figure 2) a broad range of services over the published data to the user including:

- *Virtual Schema browsing:* The user can browse the virtual relation schemas in a manner similar to browsing each of semantic categories of a large content classification system, in order to zoom the query to one or more categories he or she is interested in.
- *Keyword search:* Given a keyword query, we return a list of matching data units ranked according to their estimated relevance to the query.
- *Structured querying:* We will also provide structured query interface for user to issue structured query over one or more virtual relations. The structured query will be transformed to SQL-like query over the generic table, and be executed over it.

IV. Data units categorizer

The data units categorizer is the most important component of our system. It constructs and maintains virtual relations for representing data units published to the system of various types.

First of all, we need answer a fundamental problem — what is a qualified virtual relation in our collaborative publishing and searching system? Although the ultimate answer is subjective, we devise several objective heuristic metrics described in the following. First, we test if there is sufficient dominant attributes in the virtual relation, i.e., the attributes for which a dominant number of tuples in it have non-$null$ values. Second, we test if there is dominant tags, i.e., the tags used by a dominant number of tuples. Third, we check whether the average similarity value between the attributes of the virtual relation and the attributes of each tuple in it is above a predefined threshold.

Our hypothesis is that data units with both similar attributes and tags are likely to have similar topics and can be queried together, i.e., over the same relation schema. Therefore, data units published to the system are assigned to different virtual relations based on both their structures and their topics, i.e., we cluster the data units from two aspects. First, the data units grouped together should have similar structure — they are described with roughly similar attributes to be represented with a uniform relational schema. Second, the data units in a virtual relation should describe similar topics.

Specifically, we need to discuss several problems, which are presented in the following sections.

A. Analysis of data unit and virtual relation

In our data model, the schema of a relation is determined by the attributes of the tuples stored in it (in constrast to conventional settings where the schema is designed by domain experts a priori). Therefore, our task is to discover structure among incoming data units, and form relational schemas for groups of data units with similar topics.

Representation of features of data units: The features of a data unit include its attributes and tags. The attributes and tags of a data unit are both textual, which suggests IR related representation of the features. We employ the conventional vector space model [12] for representing both tags and attributes as *bags of words*. Each data unit is associated with two feature vectors: *attributes vector* (AV) and *tags vector* (TV). The dimensions of the vectors are the unique attributes and tags of all the data units already stored in the system. The elements are set to 1 if the corresponding attributes or tags appear in the data unit, and 0 otherwise.

Representation of features of virtual relations: From the aspect of a relation, the features of a virtual relation are its schema elements — the set of attributes, and the content of its tuples — the set of tags of its tuples. Similarly, they can also be represented with the vector space model as two feature vectors AV and TV. The elements are possibly related to two factors. The first is the popularity of the corresponding attributes or tags. Given a relation $R(U)$ and its instance I, the popularity of attribute $A \in U$ is defined as

$$popularity(A \in U) = \frac{|\{t[A] = null | t \in I\}|}{|\{t \in I\}|},$$

and the popularity of a tag $tag \in I$ is

$$popularity(tag \in I) = \frac{|\{\sigma_{match(tags,tag)>0}(I)\}|}{|\{t \in I\}|}.$$

The popularity values of attributes and tags show their importance within a virtual relation. The second factor is called the *inverse relation frequency* (IRF) values for attributes and tags in a virtual relation. The IRF value is an analog to the IDF statistics in conventional information retrieval literature. It is defined as the ratio between the number of virtual relations having the attribute/tag to the total number of virtual relations in the system. In our current implementation, we use the $popularity \cdot IRF$ value to measure the importance of an attribute/tag in a virtual relation.

Growing vocabularies: Since the data units are being added continuously, the vocabularies of both tags and attributes grow over time, and the virtual relations are also changing. This affects the corpus-level statistics such as IRF — it changes over time as new data units are added, which may have impact on term weighting and the clustering task. [6] mentions two solutions for this problem. The first approach is to use such statistics from another similar corpus. This is not practical in our context, since we cannot find an existing corpus that can be presumed to be similar to the data in our system. The second approach is to estimate the IRF statistics dynamically as data units are added to the system continuously. Initially, the values may not be accurate. However, what we care about is whether the IRF values can converge quickly as more and more data units are added to the system. We conducted an initial experiment on this issue. Figure 3 plots the changes of IRF statistics of attributes and tags measured with mean squared error as more data units are added and clustered. It can be seen from the figure the the attributes IRF statistics converges more quickly than that of the tags statistics. We believe this is because its vocabulary volume is larger than that of the attributes. Currently, we take the second approach to estimate the IRF statistics.

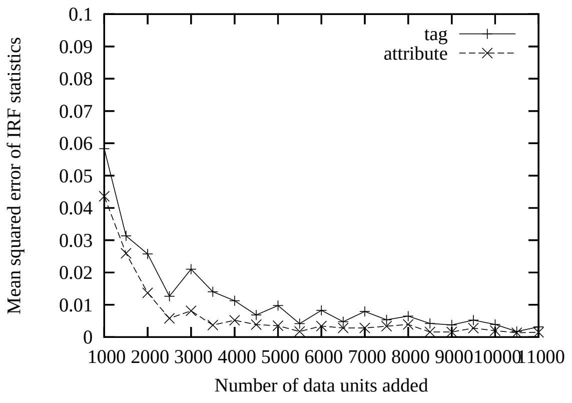


Fig. 3. Changes of IRF statistics.

Discussion: According to our experimental data[1], the distribution of the attributes and tags exhibits power-law like distribution, as shown in Figure **??** and Figure **??**. More research need to be done on how to select the proper portion of vocabularies of tags and attributes as the features of the vectors. As can be seen from the figures, there is a large portion of attributes or tags having very small frequency, which may be trivial and need not be selected as features of a virtual relation.

In addition, there are problems with the uncontrolled vocabulary of attributes and tags contributed by users, such as

[1] Our experimental data set contains over 11000 data units that are extracted from the crawled data from Google Base.

synonyms, ploysemy, superordinate, subordinate relationships, and "noises" among the attributes or tags. It is a challenging task to detect the occurrences of these problems and extract useful features from data units.

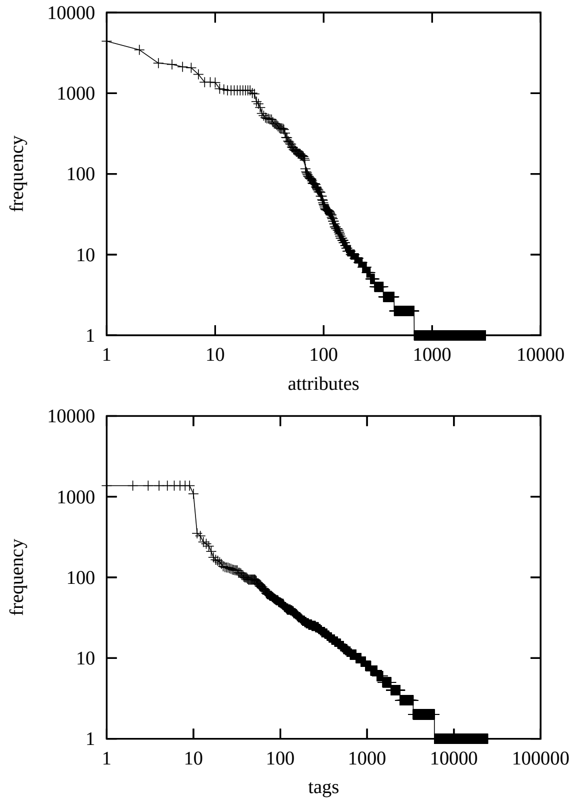


Fig. 4. Distribution of attributes and tags.

B. Similarity measure

We consider two kinds of similarities between data units u_1 and u_2. The first is the *structural similarity*, determined by the similarity between their attributes vectors. The second is the *topic similarity* based on the overlapping of their tags vectors.

Considering that the features extracted from a data unit are textual-based, which is similar to the document vectors in standard information retrieval model, we apply the cosine similarity measure widely used in IR to measure the similarities between both a pair of AV vectors and a pair of TV vectors. That is

$$sim(AV_1, AV_2) = \frac{AV_1 \cdot AV_2}{\sqrt{(AV_1 \cdot AV_1) \cdot (AV_2 \cdot AV_2)}},$$

$$sim(TV_1, TV_2) = \frac{TV_1 \cdot TV_2}{\sqrt{(TV_1 \cdot TV_1) \cdot (TV_2 \cdot TV_2)}}.$$

The similarity between a data unit and a virtual relation, and between two virtual relations are defined similarly.

C. Clustering method

Since the data units are added to the system incrementally over the time, our clustering method should also be incremental and adaptive to dynamic changes in increasing data volume and topics. We consider the incremental clustering model presented in [7].

Based on our data model, it can be described as the following. Suppose the data units published to the system are in a sequence S. A collection of up to k virtual relations is maintained such that as each new data unit $u \in S$ is presented, either it is assigned to one of the current virtual relations, or a new virtual relation is created "storing" it as the only tuple, based on the comparison of the similarities between it and all the virtual relations. If the result number of virtual relations exceeds k, two existing virtual relations are merged into one.

With two types of similarity measures between data units, we use a two-phase incremental clustering method that deploys the incremental clustering model above at each phase for our clustering task, i.e., the data units are clustered firstly based on one similarity measure, and then clustered into finer groups based on the other similarity measure. Therefore, the first problem is to decide the order of using the two kinds of similarity measures.

We observed in Figure **??** and Figure **??** that the attributes vocabulary has much smaller size than the tags vocabulary, and in general the same attributes appear more frequently in different data units than that of the tags. This suggests that we should cluster data units according to their structural similarity in the first phase, which may result in fewer clusters that are large in size. Then we further cluster data units into finer groups based on their topic similarity in the second phase. To distinguish the clusters formed in the two phases, we call the clusters formed at phase 1 as *tier 1 virtual relation*, and clusters formed at phase 2 as *tier 2 virtual relation*. A tier 1 virtual relation consists of a set of tier 2 virtual relations. Tier 2 virtual relations are the final virtual relations browsed by the user.

Basically, when a new data unit u is added to the system, the following steps may take place:

1) Extract the feature vector of u.
2) Compare the structural similarity between u and each of existing tier 1 virtual relations. Choose the virtual relation that has the maximum similarity value with u.
3) If the maximum similarity value exceeds a predefined threshold, assign the data unit to the virtual relation as a new tuple.
4) If none of the existing virtual relation has similarity value greater than a predefined threshold, create a new virtual relation and assign the new data unit to it as the only tuple.
5) If the number of virtual relations exceeds a predefined number, k_1, choose two most similar virtual relations and merge them into a new virtual relation.

When a data unit u is assigned to a tier 1 virtual relation R', it is further clustered to a tier 2 virtual relation within R' in a way similar to the steps described above, but based on the topic similarity measures.

Several parameters need be adjusted by empirical study.

1) The number of tier 1 virtual relations k_1 and the number of tier 2 virtual relations k_2.
2) Structural similarity threshold τ_s. Only when the structure similarity between a data unit and a virtual relation,

or two virtual relations is higher than τ_s, the two are considered as a match.

3) Topic similarity threshold τ_t. It determines whether the topic similarity between a data unit and a tier 2 virtual relation is high enough to assign the data unit to the tier 2 virtual relation.
4) Frequency thresholds f_h and f_l. Instead of using all vocabularies, we consider selecting the attributes or tags whose frequency is between f_l and f_h ($f_l < f_h$) as the features of a data unit.

Discussion: Currently, we assume each data unit is assigned to only 1 cluster. We may consider allowing a data unit to be assigned to multiple virtual relations. This will make the problem more complex. Also, there are issues to be considered on how to adjust the clusters when obsolete data units are deleted from the generic table.

In addition, our current approach treats the attributes and tags independently when categorizing them. It might be useful to detect the association patterns among attributes and tags in order to make the clustering process more effective.

V. Storage manager

We expect the generic table to contain thousands of attributes, and the tuples in it to be sparse – they have $null$ values in most of the attributes. This poses challenge to the physical storage scheme for the generic table. We are still evaluating various storage strategies, including the interpreted storage method for relational tuples proposed in [5]. With interpreted storage, only non-$null$ values are actually stored. Each tuple is stored as a list of non-$null$ values associated with their attribute identifiers. To handle dynamism of of data composites and huge amount of data, we need a method that is scalable over the number of attributes and data volume.

The storage manager is also in charge of maintaining the mapping between each virtual relation R_i and $AllUnits$, which is a query over the instance I of $AllUnits$, i.e., $R_i \mapsto AllUnits = q_I^i$. As data items are assigned to R_i, q_I^i need be modified as well.

When the mapping between the attributes of R_i to the attributes of $AllUnits$ is one-to-one, q_I^i is a simple projection over a subset of attributes of $AllUnits$ that matched with the attributes of R_i, respectively. There could also be the case that an attribute of R_i is mapped to multiple attributes of $AllUnits$ when the multiple attributes are detected as synonyms. In this case, q_I^i is a union over multiple projections over different combinations of the attributes of $AllUnits$ according to the attribute mappings. At current stage, we only consider simple one-to-one attribute mappings. We will investigate the issues with one-to-many mappings at a later time.

VI. Browsing and query processing

Users can look for interesting information by browsing the schemas of the virtual relations. The virtual relations are ordered decreasingly according to the number of tuples they contained for the user to browse.

The query processor supports both simple keyword queries and structured queries posed over the virtual relations. A keyword query $Q = (k_1, k_2, \cdots, k_q)$ can be posed to a particular virtual relation, or to the whole system. A fulltext inverted index is maintained for associating each of the keywords with the name of the attribute, the id of the tuple, and the virtual relation it appears in. A keyword query is processed by looking up the index and locating the positions of the relevant tuples in the generic table. Structured queries are processed by reformulating the queries posed over the virtual relations to that of the generic table according to the mappings between them, and executed over it.

VII. Conclusion and future work

We have presented our design and implementation for a collaborative publishing and searching system. We devise a data model for representing and storing the data units. We also present our approach for each system component and discuss the research challenges.

Currently, our focus in on meaningfully clustering and effective querying processing on the data units. Our ongoing work includes designing of efficient indexes on the virtual relations and data units to facilitate efficient retrieval, and efficient and adaptive query processing strategies since the less constraints imposed on the storage increased the complexity of query processing.

One of the opportunities we see in such generic storage structure is data sharing over P2P system. We intend to adopt it onto our BestPeer [11] P2P platform after we have resolved various technical issues on a centralized server.

References

[1] Delicious website. http:http://del.icio.us/.
[2] Flickr website. http://www.flickr.com/.
[3] Google base website. http://base.google.com/.
[4] Liveplasma website. http://www.liveplasma.com/.
[5] J. L. Beckmann, A. Halverson, R. Krishnamurthy, and J. F. Naughton. Extending RDBMSs to support sparse datasets using an interpreted attribute storage format. In *ICDE*, 2006.
[6] J. Callan. Document filtering with inference networks. In *SIGIR '96: Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, 1996.
[7] M. Charikar, C. Chekuri, T. Feder, and R. Motwani. Incremental clustering and dynamic information retrieval. In *Symposium on Theory of Computing*, 1997.
[8] R. Fagin, A. O. Mendelzon, and J. D. Ullman. A simplied universal relation assumption and its properties. *ACM Trans. Database Syst.*, 7(3), 1982.
[9] S. Golder and B. A. Huberman. The structure of collaborative tagging systems. Technical report, Information Dynamics Lab, HP Labs, 2005.
[10] D. Maier, J. D. Ullman, and M. Y. Vardi. On the foundations of the universal relation model. *ACM Trans. Database Syst.*, 9(2), 1984.
[11] W. S. Ng, B. C. Ooi, and K. L. Tan. BestPeer: A self-configurable peer-to-peer system. In *ICDE*, 2002. Poster Paper.
[12] G. Salton. Dynamic document processing. *Communications of the ACM*, 17(7):658–668, 1972.