Infrastructure for Efficient Exploration of Large Scale Linked Data via Contextual Tag Clouds

Xingjian Zhang, Dezhao Song, Sambhawa Priya, and Jeff Heflin

Dept. of Computer Science and Engineering, Lehigh University
19 Memorial Drive West, Bethlehem, PA 18015, USA
{xiz307,des308,sps210,jeh3}@lehigh.edu

Abstract. In this paper we present the infrastructure of the contextual tag cloud system which can execute large volumes of queries about the number of instances that use particular ontological terms. The contextual tag cloud system is a novel application that helps users explore a large scale RDF dataset: the tags are ontological terms (classes and properties), the context is a set of tags that defines a subset of instances, and the font sizes reflect the number of instances that use each tag. It visualizes the patterns of instances specified by the context a user constructs. Given a request with a specific context, the system needs to quickly find what other tags the instances in the context use, and how many instances in the context use each tag. The key question we answer in this paper is how to scale to Linked Data; in particular we use a dataset with 1.4 billion triples and over 380,000 tags. This is complicated by the fact that the calculation should, when directed by the user, consider the entailment of taxonomic and/or domain/range axioms in the ontology. We combine a scalable preprocessing approach with a specially-constructed inverted index and use three approaches to prune unnecessary counts for faster intersection computations. We compare our system with a state-of-the-art triple store, examine how pruning rules interact with inference and analyze our design choices.

Keywords: Linked Data; Tag Cloud; Semantic Data Exploration; Scalability

1 Introduction

We present the contextual tag cloud system[1] as an attempt to address the following questions: How can we help casual users explore the Linked Open Data (LOD) cloud? Can we provide a more detailed summary of linkages beyond the LOD cloud diagram[2]? Can we help data providers find potential errors or missing links in a multi-source dataset of mixed quality? There are two aspects of a dataset: the ontological terms (classes and properties) and the instances; and correspondingly, there are two types of linkages: ontological alignment and

[1] Contextual Tag Cloud Browser. http://gimli.cse.lehigh.edu:8080/btc/

[2] The Linking Open Data cloud diagram. http://lod-cloud.net/

`owl:sameAs` links between instances. We allow the user to specify a *context* as a combination of ontological terms, and then visualize the degree of overlap between this context and all other terms. The context can be thought of as a class expression in description logic, but is significantly simplified for usability reasons. The overlap is the intersection of the context class and any other term. An appropriate visualization of these counts can reflect the patterns of co-occurrence of ontological terms as used in the instance data.

We build on the idea of a contextualized tag cloud system. In analogy to traditional Web 2.0 tag cloud systems, an instance is like a web document or photo, but is "tagged" with formal ontological classes, as opposed to folksonomies. Thus, we simply use "tags" as another name for the categories of instances. We extend the expressiveness and treat classes, properties and inverse properties as tags that are assigned to any instances using these ontological terms in their triples. The font sizes in the tag cloud reflect the number of matching instances for each tag. To explore the data, users can select a set of tags to form a context and the displayed tags are resized to indicate intersection with this context. Note, this system is neither an information retrieval system nor a SPARQL query engine, instead it is designed for exploration and pattern discovery.

With any uncurated dataset, one must maintain a healthy skepticism towards all axioms. Although materialization can lead to many interesting facts, a single erroneous axiom could generate thousands of errors. Rather than attempting to guess which axioms are worthwhile, our system supports multiple levels of inference; and at any time a user can view tag clouds with the same context under different entailment regimes, which helps users understand the dataset better and helps data providers investigate the errors in the dataset.

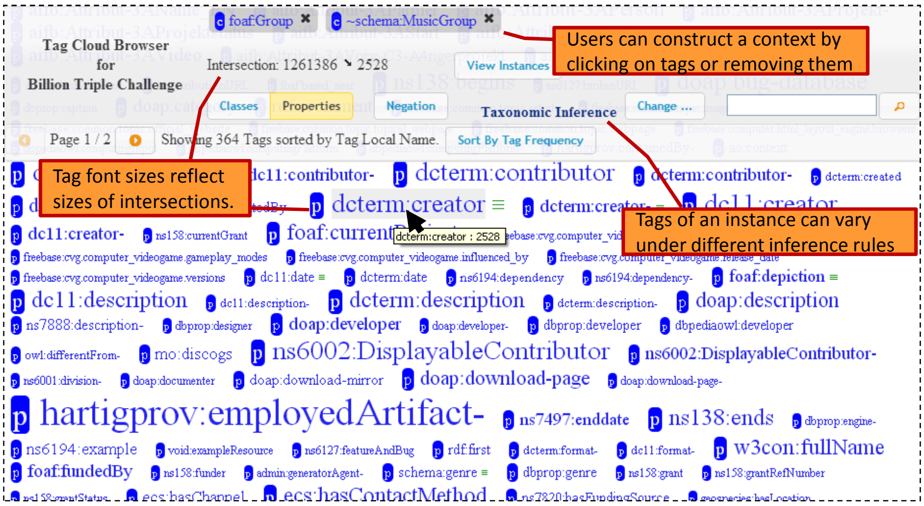
These simple but powerful interface concepts propelled the Contextual Tag Cloud Browser to win the Billion Triples Track of the 2012 Semantic Web Challenge[3]. Our initial version of the system [17] was used on DBPedia data [3]. For the Semantic Web Challenge, we added features and loaded the entire 2012 BTC dataset. This complex dataset contains 1.4 billion triples, from which we extract 198.6M unique instances, and assign more than 380K tags to these instances. This multi-source, large-scale dataset brings us challenges in achieving acceptable performance and user-interface design. Although we believe the user interface provides a convenient tool for exploring a Linked Data dataset, the focus of this paper is presenting novel approaches for efficient and scalable computation over noisy data with tremendous diversity.

The contributions of this paper are: (1) We propose using an inverted index to speed up a special kind of query, namely querying the intersection of generalized classes, and propose a scalable approach to preprocess it; (2) Some special cases of these queries can be answered without accessing the index, we propose three approaches to prune unnecessary queries and analyze alternative preprocessing approaches; (3) We develop formulae for supporting the first problems with multi-level inference and discuss our decision to materialize entailments and an efficient mechanism to store the results of these entailments. Although this

[3] SWC 2012 Winners. `http://challenge.semanticweb.org/2012/winners.html`

paper focuses on a very specific application, we believe scalable computation of conditional distributions can be applied to statistic based algorithms such as association rule learning. The rest of the paper is organized as follows: we first briefly describe the use cases of the tag cloud system and formally define the problem; then we discuss the preprocessing and online computation and how we support multi-level inference; after that we provide some experimental results of the system; then we compare with related works; and lastly we conclude.

2 The Problem: Use Case and Formal Definition



Users can construct a context by clicking on tags or removing them

Tag font sizes reflect sizes of intersections.

Tags of an instance can vary under different inference rules

Fig. 1. Property Tag Cloud with context `foaf:Group` and ~`schema:MusicGroup`.

Initially, the system shows a tag cloud with no context tags selected, and the tags in the cloud reflect the number of instances related to each tag. If a tag is clicked, it will be added to the current context, and then a new tag cloud will be shown for the updated context. A user can add/remove any tags to/from the context, and explore any dynamically defined types of instances specified by the context. Then in the resulting tag cloud, the font size for each tag reflects the number of instances possessing the tag within the type specified by the contexts. Mathematically, this contextual tag cloud actually reveals the **conditional distribution** of the data: the probability that an instance has a tag given that it is an instance of the user-defined type. For example in Fig. 1, the property tag cloud shows us the degree to which instances of `foaf:Group` that are not in `schema:MusicGroup` are used with specific properties.

This kind of pattern visualization helps users learn about the dataset for different purposes. For example, large tags indicate frequent co-occurrence and can be used to form a SPARQL query spanning diverse, multiple, linked data sources that is most likely to return results; by focusing on the smaller tags, users can investigate rare combinations, and by drilling into the data determine if these are unusual facts or the product of data errors, such as incorrect `owl:sameAs` links. Additionally the user can dynamically change which entailment regime will be used to generate the tag cloud, thereby getting a big picture view of the impact of entailment on the data. This feature can be used to track down schema errors such as incorrect `rdfs:domain` statements.

An important aspect of the user interface is responsiveness. Ideally, each new tag cloud should be generated in under one second, or users will quickly doubt the system and/or become bored. Achieving this goal is particularly challenging since the dataset contains billions of triples with hundreds of thousands of ontological tags. In addition to an interface design that ensures the user is presented with partial results as quickly as possible, we carefully designed an infrastructure that is optimized for our unique form of queries. Before we describe our approach, we now formalize the computation problem.

Given a RDF dataset, an entailment regime R defines what kind of entailment rules will be applied to the explicit triples. In our implementation, we have two specific sets of rules: R_{Sub} for sub/equivalent class/property entailment (`rdfs5`, `rdfs7`, `rdfs9` and `rdfs11` [4]); and R_{DR} for property domain/range entailment (`rdfs2`, `rdfs3`). We also support the combination of these two sets, leading to four distinct entailment regimes $\mathcal{R} = \{\emptyset, R_{Sub}, R_{DR}, R_{Sub} \cup R_{DR}\}$.

Let $\mathcal{I}$ be the set of all the instances, and $\mathcal{T}$ be the set of all possible tags assigned to instances in the dataset. Given R, we define a function $\text{Tags}_R : \mathcal{I} \rightarrow 2^{\mathcal{T}}$ that returns all the tags assigned to the given instance under R-inference closure. For $i \in \mathcal{I}$ we assign three types of tags: (1) **Class** C, if $\langle i$, `rdf:type`, $C\rangle$ is entailed under R. (2) **Property** p, if $\exists j \in \mathcal{I}$, s.t. $\langle i, p, j\rangle$ is entailed. (3) **Inverse Property** $p-$, if $\exists j \in \mathcal{I}$, $\langle j, p, i\rangle$ is entailed. Note under monotonic logic, $R_1 \subseteq R_2 \Rightarrow \text{Tags}_{R_1}(i) \subseteq \text{Tags}_{R_2}(i)$. The function $\text{Inst}_R : 2^{\mathcal{T}} \rightarrow 2^{\mathcal{I}}$ returns the set of all instances assigned the given set of tags. For convenience, we define the frequency of a set of tags T as $f_R(T) = |\text{Inst}_R(T)|$.

We can generalize various entailment rules into tag subsumptions. Tag t_1 is a sub tag of tag t_2 if and only if the entailment regime requires $\text{Inst}_R(\{t_1\}) \subseteq \text{Inst}_R(\{t_2\})$. This sub tag relation includes RDF subclasses/subproperties plus the ones entailed by the domain/range axioms: If $\langle p$, `rdfs:domain`, $C\rangle$ and $\langle p$, `rdfs:range`, $D\rangle$, then p is a sub tag of C and $p-$ is a sub tag of D. We compute the full closure on tag subsumptions for each inference regime.

A context is an expression of tags dynamically constructed by a user. In our implementation, it is the intersections of any number of tags or their negations. A **Negation Tag** $\sim t$ is virtually assigned to an instance i, if $t \notin \text{Tags}_R(i)$. Note that the semantics are based on negation-as-failure. We argue that this is the correct semantics for a system where what is not said is sometimes as important as what is said. Thus a context with $\{t_1, \ldots, t_n, \sim s_1, \ldots, \sim s_m\}$ actually defines a subset of instances: $\text{Inst}_R(\{t_1, \ldots, t_n\}) - \bigcup_{x=1,\ldots,m} \text{Inst}_R(\{s_x\})$. For a given

[4] RDFS Entailment Rules: `http://www.w3.org/TR/rdf-mt/#RDFSRules`

context and entailment regime R, the system shows all the tags used by any instance in the subset specified by the context, and the size of each tag reflects the number of instances having this tag within the subset.

Due to limited space, we omit the subtle details required to process negation tags for the remainder of this paper. This allows us to present a simplified exposition where a context $T \subset \mathcal{T}$ is a set of tags, and the instances specified by the context is $\text{Inst}_R(T)$. For more details, please refer to our technical report [18].

Since our goal is to display the frequency of all tags given a context T, our main challenge is to compute $f_R(\{t\} \cup T)$ for $\forall t \in \mathcal{T}$ efficiently. There are two ways to approach this problem: (1) ensure efficient calculation of $f_R(T)$ for any T; and (2) prune unnecessary calls of $f_R(\{t\} \cup T)$. To achieve this, we need to correctly structure the repository and develop an efficient preprocessing step. In the following section we will solve these problems for the situation where there is only a single set of inference rules R. Then we will discuss how to "infer" relations between tags and instances, and how to determine co-occurrence between tags under tag inference.

3 Preprocessing

Our previous experiments [17] showed that an RDBMS with decomposed storage model [1, 11] is not as efficient as using an Information Retrieval (IR) style index for this specific application purpose, both in terms of load time (8X slower) and online query time (18X slower). Therefore we extend our IR approach, but meanwhile add more steps to deal with the BTC dataset.

Our preprocessing is shown in Figure 2, where the dashed boxes are input or intermediate data and the solid ones are data results for the online system. First, we parse the raw data and categorize triples into three files: the ontology file which includes specific properties (e.g., `rdfs:subClassOf`) or classes (e.g., `owl:Class`), the `owl:sameAs` (instance equivalence statements) file, and the file of remaining instance triples. This step can be skipped if the ontology and sameAs files are provided separately. The ontology is processed into a closure set of sub-tag axioms for the given entailment regime (or regimes); the closure is then responsible for two functions: $\text{sub}_R(t)$ and $\text{super}_R(t)$ which respectively return the sets of sub/super tags of tag t under inference R. We also use the well-known union-find algorithm to compute the closure for `owl:sameAs` statements, and pick a canonical id for each `owl:sameAs` cluster.

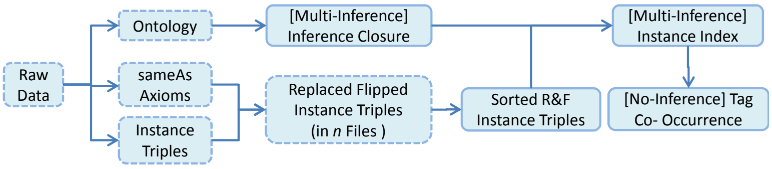

Raw Data
Ontology
sameAs Axioms
Instance Triples
[Multi-Inference] Inference Closure
Replaced Flipped Instance Triples (in *n* Files)
Sorted R&F Instance Triples
[Multi-Inference] Instance Index
[No-Inference] Tag Co- Occurrence

Fig. 2. Preprocessing for the tag cloud system

Then for the instance triples, we replace each instance with its `owl:sameAs` canonical id (if any). If the object of the triple is also an instance, we flip the triple and add it to the intermediate file, i.e., if the triple is $\langle i, p, j\rangle$, the flipped one is $\langle j, p-, i\rangle$. By this means, we can find all the tags (particularly inverse property tags) of an instance i by simply looking at the triples with i as a subject. In order to index an instance, we need to first group all of its triples together. To do this, we first output the triples into n files based on the hashcode of their subjects, so that we keep the information of an instance in the same file while making each file relatively small. Then we use merge sort on each "replaced and flipped" file, so that triples with the same subject instance are clustered together. Note that by splitting the triples into n files, we gain benefits from two sides: (1) sorting each file becomes faster (and since we only need to group triples with the same subject, we do not need to merge the sorted files); (2) we can sort in parallel (either with multiple machines or with multiple threads). We use these sorted files together with the given inference closure to build an inverted index of the instances.

The inverted index is built with tags as indexing terms and each tag has a sorted posting list of instances with that tag. This means given a "type" defined by a set of tags we can quickly find all the instances by doing an intersection over the posting lists. Also, since we use negation as failure, we do not need to index negation tags; their size can be calculated from its complementary tag. i.e. $f_R(\{\sim t\} \cup T) = f_R(T) - f_R(\{t\} \cup T)$. Meanwhile we add other fields such as labels of instances, sameAs sets, file pointers to the raw file, etc. to facilitate other features in our tag cloud system.

To help prune unnecessary tags when computing the conditional distribution of tags under any given context T, we precompute the **Co-occurrence Matrix** for all the tags. Define M_R as a $|\mathcal{T}| \times |\mathcal{T}|$ symmetric boolean matrix, where $M_R(x, y)$ denotes whether tags t_x and t_y co-occur, i.e. $M_R(x, y) = (f_R(\{t_x, t_y\}) > 0)$. There are three ways to generate M_R.

1. **Traverse all the instances.** For each instance $i \in \mathcal{I}$, we get all of its tags $\text{Tags}_R(i)$, for any pair of tags $(t_x, t_y) \in \text{Tags}_R(i) \times \text{Tags}_R(i)$, set $M_R(x, y)$.
2. **Traverse pairs of tags.** For any pair of tags $(t_a, t_b) \in \mathcal{T} \times \mathcal{T}$, if $f_R(\{t_a, t_b\}) > 0$, set $M_R(x, y)$.
3. **Traverse tag instances.** For each tag $t_x \in \mathcal{T}$, we get all of its instances $\text{Inst}_R(\{t_x\})$, and then set occurrences for all tags in them. For $i \in \text{Inst}_R(t_x)$, for any tag $t_y \in \text{Tags}_R(i)$, set $M_R(x, y)$.

We can roughly estimate the execution time of each method from how much index access (the functions Tags_R, f_R, and Inst_R) is needed. Assume on average a tag has d instances and an instance has e tags. The cost of $\text{Inst}_R(\{t_x, t_y\})$ (or $f_R(\{t_x, t_y\})$) is estimated as $c_1 d$, because the intersection needs to simultaneously walk through both sorted posting lists. The cost of $\text{Tags}_R(i)$ is estimated as $c_2 e$. Here, c_1, c_2 are constants given the dataset and the environment. Roughly speaking, the first method has $|\mathcal{I}|$ iterations and takes $|\mathcal{I}|c_2 e$; the second has $|\mathcal{T}|^2/2$ iterations and takes $c_1 d|\mathcal{T}|^2/2$; and the third has $d|\mathcal{T}|$ iterations and takes $c_2 ed|\mathcal{T}|$.

There is one problem with the estimations above: we ignored the cost of setting M. For the second and the third approach, they both only need to set each $M_R(x, y)$ once; the first approach however can repeatedly set the same cell. What is even worse, for a large scale dataset, we might be unable to have the full matrix in memory, and thus updating random cells becomes more costly. In contrast, the third approach calculates cells row by row, and both the second and the third approach can stream out the results since each cell is set at most once. When choosing between the second and the third approach, we pick the third one if the ratio $r = \frac{c_1 d|\mathcal{T}|^2/2}{c_2 ed|\mathcal{T}|} = \frac{c_1|\mathcal{T}|}{2c_2 e} > 1$. Note both c_1 and c_2 can be easily estimated by experiment, and c_2 is usually one to two orders of magnitude larger than c_1. In general, if the size of all the tags is small enough to hold the full matrix in memory, then use the first approach; otherwise, if we find in the dataset that each instance usually uses a very small portion of all the tags (e.g. less than 1%), the third approach is preferred than the second. In a multi-source cross-domain dataset such as the BTC dataset, instances usually have very few tags from other domains, e.g. a musician instance will seldom use tags from domains like e-Government or life sciences; thus we use third approach.

This matrix provides a function for each tag t_x to return all the tags that co-occur with it in at least one instance. i.e. $\mathrm{CO}_R(t_x) = \{t_y | M_R(x, y) = 1\}$. We shall discuss the significance of this function next.

4 Online Computation

Given a context T and entailment regime R, the online computation will return all the $f_R(\{t\} \cup T)$ for every tag t. With our index, we can simply issue an IR query for each t that counts all the instances with all tags in T and t, which is getting the number of total hits for a boolean `AND` query. Note that the underlying system compares the posting lists of all tags in the query, and because T is the common part among this series of queries, the intersected posting list can be shared among queries. Thus increasing $|T|$, i.e., the size of the context, may simplify the queries by generating a shorter posting list for T. A quality IR system can answer a count query within a few milliseconds, but since we have hundreds of thousands of tags, we need to focus on how to reduce the number of queries.

There are two special cases of the f_R results: (1) $f_R(\{t\} \cup T) = f_R(T)$; and (2) $f_R(\{t\} \cup T) = 0$. The question is whether we can know the results without issuing f_R queries. For the first case, if t is a super tag of any tag in T, i.e. $\exists t' \in T, t \in \mathrm{super}_R(t')$, adding t to T does not change the instance set and thus does not change f_R. For the second case, ideally we want to skip every tag with an empty intersection without issuing a query. For convenience, we define $Z_R(T) = \{z | f_R(\{z\} \cup T) = 0\}$. Let CL be the candidate list of tags whose queries are finally issued. We propose three different pruning approaches to make CL as short as possible.

1. **Use the Co-occurrence Matrix (M).** Given T, $\bigcap_{t' \in T} \mathrm{CO}_R(t')$ has (and not necessarily only has) all the tags $\{t | f_R(\{t\} \cup T) > 0\}$. When $|T| = 1$, it

returns only the co-occurring tags and prunes all the $Z_R(T)$. When $|T| > 1$, it returns a super set of the co-occurred tags, because the returned tags are only known to pairwise co-occur with any tag in T, but are not guaranteed to co-occur with all tags in T in the same instance.

2. **Use the previous tag cloud cache (C).** Since $\text{Inst}_R(\{t\} \cup T) \subseteq \text{Inst}_R(T)$, the set of co-occurred tags given context $\{t\} \cup T$ is also a subset of that given T. Thus if we cache the previous tag cloud, which has the same context T except for the most recently added tag, we can get another super set of the co-occurred tags for context T. This relies on the tag cloud application scenario: it is very likely that the current request is from a user adding a new tag to the context. However we believe it can be applied to any scenarios involving a depth-first search of the context space.
3. **Dynamic update (D).** When computing $f_R(\{t\} \cup T)$ for all the candidate tags from the above two approaches, if we find $f_R(\{t_x\} \cup T) = 0$, we know $\forall t_y \in sub_R(t_x), t_y \in Z_R(T)$, and these tags will be ignored in further computation. This approach can be optimized if we sort the list of tags such that sub tags always follow super tags. However, our tag cloud system does not use this optimization because it needs to stream results alphabetically.

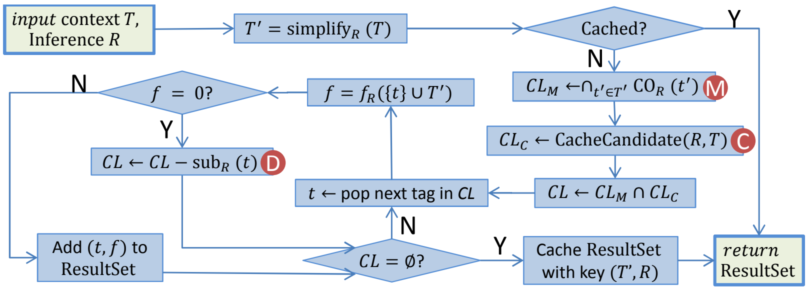


Fig. 3. Pruning for Online Computation

The online computation works as shown in Fig. 3, where the pruning steps are marked with red circles. First, the input context T will be simplified (under R-Inference) to its semantic-equivalent T' so that any redundant tags will be removed (e.g., if $T = \{t_1, t_2\}$ and t_1 is a super tag of t_2 then $T' = \{t_2\}$) and any equivalent tag will be changed deterministically to a representative tag. Then the system checks whether this semantic-equivalent request has been kept in cache for direct output. If not, the system will get candidate lists CL_M from the first approach using T' and CL_C from the second approach using T. Then we use the intersection $CL = CL_M \cap CL_C$ as the candidate list for queries and keep updating it using the third approach. It is easy to prove that using simplified T' in the first approach will get the same candidate tags as using T. Given $T = \{t_1, t_2\}$ where t_1 is a super tag of t_2, we can see $\text{CO}_R(t_1) \cap \text{CO}_R(t_2) = \text{CO}_R(t_2)$ since $\text{CO}_R(t_1) \supseteq \text{CO}_R(t_2)$. However by removing super tags, we can avoid unnecessary intersection of lists when computing the candidates. On the other hand, the cache approach needs the original T in order to get the previous context; subsequently, this previous context is simplified for cache lookup.

5 Supporting Different Entailment Regimes

From the raw dataset, we get only $\text{Tags}_\emptyset$, the tags of each instance with no inference applied. In order to implement Tags_R, Inst_R and CO_R for different R, we can either materialize them so that they serve as independent repositories; or we can always do the inference on-the-fly. We first discuss how to represent the three functions under R by combining the $R = \emptyset$ versions (i.e., with no inference) with the tag subsumption functions super_R and sub_R. After that we will discuss the design choice regarding materialization.

By adding inference, an instance will be assigned with the super tags of its explicit tags, and a tag will be assigned to all instances of its sub tags. i.e.

$$\text{Tags}_R(i) = \bigcup_{t' \in \text{Tags}_\emptyset(i)} \text{super}_R(t') \tag{1}$$

$$\text{Inst}_R(T) = \bigcap_{t \in T} \bigcup_{t' \in \text{sub}_R(t)} \text{Inst}_\emptyset(t') \tag{2}$$

From Eq. (1), we know that

$$t \in \text{Tags}_R(i) \Leftrightarrow \exists t' \in \text{sub}_R(t), t' \in \text{Tags}_\emptyset(i) \tag{3}$$

If tag s co-occurs with tag t under R,

$$\begin{aligned} s \in \text{CO}_R(t) &\Leftrightarrow \exists i \in \mathcal{I}, s \in \text{Tags}_R(i) \wedge t \in \text{Tags}_R(i) \\ &\Leftrightarrow \exists i \in \mathcal{I}, \exists s_x \in \text{sub}_R(s), \exists t_y \in \text{sub}_R(t), s_x \in \text{Tags}_\emptyset(i) \wedge t_y \in \text{Tags}_\emptyset(i) \\ &\Leftrightarrow \exists s_x \in \text{sub}_R(s), \exists t_y \in \text{sub}_R(t), s_x \in \text{CO}_\emptyset(t_y) \end{aligned} \tag{4}$$

For convenience, we define

$$\text{super}_R^\cup(T) = \bigcup_{t' \in T} \text{super}_R(t') = \{t | \exists t' \in T, t \in \text{super}_R(t')\} \tag{5}$$

And similarly,

$$\text{CO}_R^\cup(T) = \bigcup_{t' \in T} \text{CO}_R(t') = \{t | \exists t' \in T, t \in \text{CO}_R(t')\} \tag{6}$$

Then we compute $\text{CO}_R(t)$ from Eq. (4).

$$\begin{aligned} \text{CO}_R(t) &= \{s | \exists s_x \in \text{sub}_R(s), \exists t_y \in \text{sub}_R(t), s_x \in \text{CO}_\emptyset(t_y)\} \\ &= \{s | \exists t_y \in \text{sub}_R(t), \exists s_x \in \text{CO}_\emptyset(t_y), s \in \text{super}_R(s_x)\} \\ &= \text{super}_R^\cup(\{s_x | \exists t_y \in \text{sub}_R(t), s_x \in \text{CO}_\emptyset(t_y)\}) \\ &= \text{super}_R^\cup(\text{CO}_\emptyset^\cup(\text{sub}_R(t))) \end{aligned} \tag{7}$$

In our implementation, as shown in Fig. 2, we materialize Tags_R for all 4 entailment regimes, thus we do not need to compute Eq. (2) for online computation. However we only precompute $\text{CO}_\emptyset$ and use Eq. (7) at online computation. We made our design choices based on two reasons. First, How much slower will it be if not materialized? Both Eq. (2) and (7) include union and intersection of sets or posting lists, however the lists of instances are usually much larger and using Eq. (2) significantly increases the execution time compared to the materialized index. Second, How important is the runtime performance? As in our scenario, for each tag cloud (or conditional distribution) given T, CO_R is only called once, however Inst_R is called for each tag from the candidate set.

Also note Eq. (7) can be used for either online computation of CO_R or precomputation if it is materialized. Building the co-occurrence matrix M_R is a time consuming step (see Fig. 4). We should avoid repeating it four times for four inference regimes. Instead, we only need to build $M_\emptyset$, which is the easiest because each instance has the minimal number of tags, and the co-occurrence for all the other inference regimes can be computed based on Eq. (7).

6 Experiments

Our system is implemented in Java and we conducted all experiments on a RedHat machine with a 12-core Intel 2.8 GHz processor and 40 GB memory.

In order to test the performance of our preprocessing approach, we apply it to all five subsets of the BTC 2012 dataset, as well as the full dataset. The statistics are listed in Table 1.

Table 1. Statistics of Triples in the subsets of BTC 2012 dataset

Set Name	Total	Ontology Triples	SameAs Triples	Instance Triples
`rest`	22 M	54.7 K	734 K	∼22 M
`freebase`	101 M	0	897 K	92 M
`dbpedia`	198 M	1.8 K	22,818 K	175 M
`timbl`	205 M	1,260.1 K	340 K	203 M
`datahub`	910 M	466.0 K	4,490 K	905 M
full set	1,437 M	1,782.6 K	37,357 K	1,397 M

Fig. 4 illustrates how long each step of preprocessing takes for each subset. The Multi-Inference step is not included in the figure since it is too short (41s for the full set) compared with other steps. In general the sorting step and the steps that involve a full scan of the dataset, such as Replace/Flip and index, are the most substantial. Each step is related to certain factors of the dataset provided in Table 1. E.g. the time for inference is related to the number of tag subsumption axioms, which is correlated with the number of ontology triples; the time for union-find on sameAs is related to the number of SameAs triples; and most of the other steps are related to the number of instance triples. Despite the differences in the portions of different kinds of triples, we also plot the time/space for datasets against their numbers of total triples in Fig. 5, which shows the scalability of our preprocessing approach. The reported disk space includes both the index and the no-inference co-occurrence matrix $(M_\emptyset)$, and is dominated by the index, which usually takes $> 90\%$. We can see the time is quite linear with the total number of triples, because most of the major steps are linear w.r.t. the number of triples. The space however is slightly less correlated to the total number of triples, since many different triples might only contribute to a single tag in the index. For example, there might be 1000 triples saying a `foaf:Person` `foaf:knows` 1000 different people, however these triples only contribute a single property tag to this person. This is exactly what happens in the `timbl` subset, and explains why we see `timbl` has slightly more triples than `dbpedia` but needs less time/space.

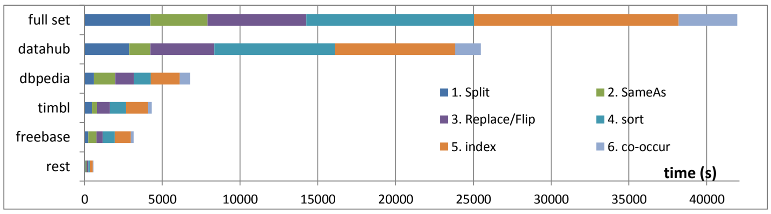


Fig. 4. Time for steps of preprocessing various datasets

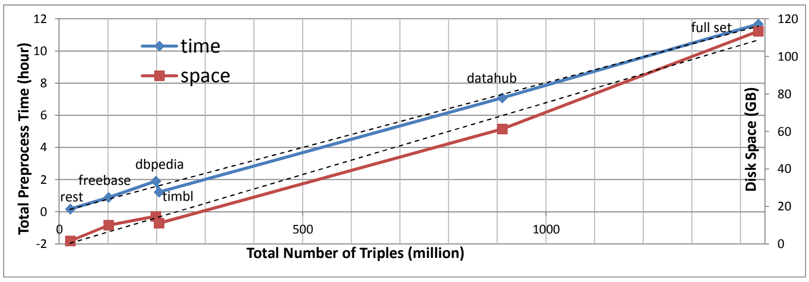


Fig. 5. Preprocessing: Time/Space - Total Triples

We then test the response time of $f_R(\{t\} \cup T)$ queries, i.e. how long it takes to count the instances of tag t with context T by querying the index. To ensure a random but meaningful context T, i.e. $\mathrm{Inst}_R(T) \neq \emptyset$, we randomly pick an instance i and get a subset (size of 6) from its tags $\mathrm{Tags}_\emptyset(i)$ as $[t_{i,1}, t_{i,2}, \ldots, t_{i,6}]$. Thus the six tags in this array are known to co-occur under all entailment regimes. We generate 100 such arrays using different i. Additionally, we pick a set S of 10000 random tags. Starting from[5] $k = 1 \ldots 6$, we use the first k tags in the arrays as contexts T, and we measure the average time of $f_R(\{s\} \cup T)$ for all $s \in S$. While S might overlap with some T, it does not impact the query time since we issued the same f_R queries without removing redundant query terms. By doing this, we can compare the average query time for different contexts T because they are intersected with the same tags; and we can compare the difference when adding more tags to contexts because as k increases, each array will provide a more "strict" context then before. We also change $R = \emptyset, R_{Sub}, R_{DR}, R_{Sub} \cup R_{DR}$ to examine the impact of different inference. The average time per 10K queries grouped by $|T|$ is shown in Fig. 6. In average, it takes 0.6∼0.7 milliseconds for a single f_R query. The time slightly increases (sub-linear) when we add more tags to context. It takes longer if R has more inference rules due to longer posting lists of tags in the index. As we expect, since there are fewer tags added to each instance from domain/range inference, we find the curves for R_{DR} and $\emptyset$ are close, while R_{Sub} and $R_{Sub} \cup R_{DR}$ are nearly identical.

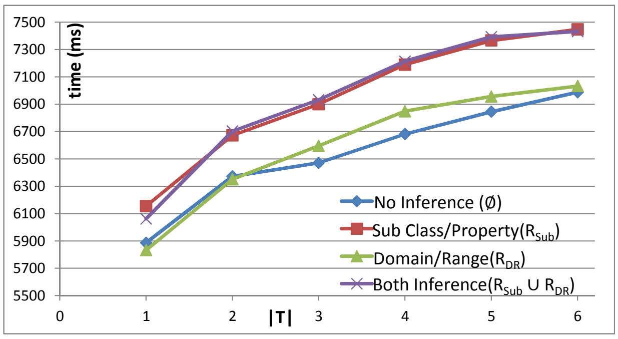


Fig. 6. Average time for 10K queries as context T grows for each entailment regime.

A reasonable question is whether a high-performance triple store could be used as a backend for our system. To answer this question, we compare the response time of this specific kind of queries with RDF-3X [10] a state-of-the-art SPARQL engine that "outperforms its previously best systems by a large margin". It takes 9 hours and 11 minutes to load the full BTC dataset into RDF-3X. Note that this loading does not include any kind of inference, sameAs closure/replacement, nor co-occurrence computation as we do in our preprocessing. Similar to the previous experiment, for context size $|T| = 1, \ldots, 5$, we randomly pick 50 (10 of each) contexts, and this time we measure how long it takes for both systems to compute the full contextual tag cloud without pruning. i.e. for a given T, we compute $f_\emptyset(\{t\} \cup T)$ for $\forall t \in \mathcal{T}$. We use $R = \emptyset$ because RDF-3X does not explicitly do any inference. The comparison results are shown

[5] The initial tag cloud ($|T|$=0) is precomputed and cached, thus we do not test it here.

in Table 2. In addition to the average execution time of both systems, we also list the Average/Maximum/Minimum Differences, which shows how much faster our system is compared to RDF-3X, with respect to an average query, its best query and its worst query. Note, the times in this table are longer than those in Fig. 6, because we are issuing ∼380K queries as opposed to 10K. It is clear that our system always outperforms RDF-3X. Averaging across all queries in our test set, our system is 10 times faster than RDF-3X. The differences are more pronounced when $|T|$ increases, although both systems have a sub-linear increase in query execution time as $|T|$ increases. There are two outliers of the Max/Min trends. When $|T| = 1$, the Max Diff. occurs when $f_\emptyset(T) = 49,584,018$, which is the largest set of instances specified by the context in our test set. When $|T| = 5$, the Min Diff. occurs when $f_\emptyset(T) = 143$, which is the smallest set of instances specified by the context in our test set. It is possible that the smaller sizes of instances specified by the context lead to more efficient joins in RDF-3X, allowing it to approach our system's performance. The key point to recognize here is that one-size-fits-all triple stores are not always the best solution for scalable applications. By choosing a carefully constrained user interaction method, we are able to design a specialized infrastructure that can meet our performance requirements. That said, we posit that the systems capable of performing voluminous tag intersections can be used not just for supporting user interfaces, but for data mining and anomaly detection as well.

Table 2. Comparison on Time Cost for Computing Full Tag Cloud (No Pruning)

$\|T\|$	Avg. Time Ours	Avg. Time RDF-3X	Avg. Diff.	Max Diff.	Min Diff.
1	65.8 s	887.6 s	13.5 X	93.2 X	1.71 X
2	84.9 s	516.7 s	6.09 X	15.6 X	2.87 X
3	90.7 s	721.2 s	7.95 X	20.6 X	4.56 X
4	92.8 s	1030.8 s	11.1 X	30.8 X	6.24 X
5	110.3 s	1359.7 s	12.3 X	33.4 X	4.44 X
All	88.9 s	903.2 s	10.2 X	93.2 X	1.71 X

We also test how well our system does for pruning candidate tags under the most complex inference $R = R_{Sub} \cup R_{DR}$. Using the approach above, we generate 100 arrays of length 6 from $\text{Tags}_R(i)$, by changing the length of sub arrays we get 600 random T. As we discussed in the previous section, there are three approaches: by co-occurrence matrix (**M**), by previous cache (**C**), or by dynamic update (**D**). By each combination of approaches, we can count how many f_R queries are finally issued, and see how many queries are pruned. Note there is always some pruning due to super tags of tags in contexts. When using approach C, we always assume the previous cache is available.

The average number of pruned tags is shown in Fig. 7. There are $|\mathcal{T}|$ =389K tags in total however most tags only co-occur with a few other tags. Pruning usually saves us unnecessary queries. We can see when $|T|$ increases any approach will generally prune more tags because more tags in T means a more

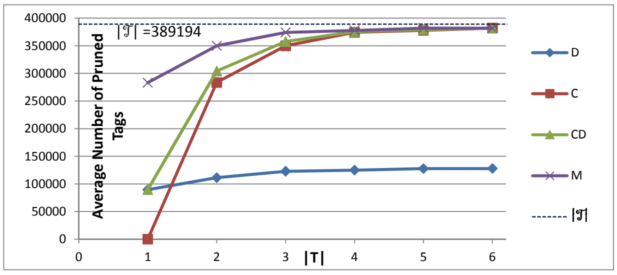


Fig. 7. Average Number of Pruned Tags

constrained context. Among the three approaches, M in average prunes more tags, and enabling the other two approaches with M only provides less than 1% more pruning (thus we do not show the overlapping curves for combinations MC, MD and MCD). This justifies the preprocessing for the co-occurrence matrix. C also has good pruning except that when $|T| = 1$, the cache of $|T| = 0$ is a list of every tag and C will not help. However, in the tag cloud scenarios, $|T| = 1$ is important as it will decide the response after the user's first click. Also in practice, the history cache might not always be available (e.g. a user adds t_1, t_2, t_3 and then removes t_2). So its availability is a concern although it requires no preprocessing. The time cost for CO_R is not a key concern to our system. The average time for the above test set is $1.1s$ with all approaches enabled. However running this pruning saves ∼300K f_R queries or in average $0.6ms \times 300K = 180s$ for each tag cloud. For the above 600 T, we have an average time of $8.8s$ per tag cloud, with max of $48.8s$. Thanks to the paging and streaming features in our interface design, the first 200 tags in the tag cloud page almost always show within 2 seconds, which we consider an acceptable responsiveness.

7 Related Work

To the best of our knowledge, we have not seen any other works like the contextual tag cloud system, nor papers focusing on optimization for the specific kind of query and resolving related problems. To compare with general purpose triple stores, Rohloff et al. [12] present a comparison of scalability performance of various triple store technologies using the LUBM benchmark [8], and reported that Sesame [4] was the most scalable: It loads around 850M triples in about 12 hours, but it takes more than 5 hours to answer LUBM Query 14, which, similar to our task, requests the instances of a class. Sakr and Al-Naymat [13] survey RDF data stores based on relational databases and classify them into three categories: (1) each triple is stored directly in a three-column table, (2) multiple properties are modeled as n-ary table columns for the same subject, and (3) each property has a binary table. Abadi, et al. [2] explore the trade-off and state the third category is superior to the others on queries. However our previous experiments [17] show using an inverted index is much faster for the queries that count instances of intersections of classes/properties. In this paper we continue to compare our inverted index approach with the state-of-the-art RDF store RDF-3X [10]. The difference in the experiments indicates that a general purpose SPARQL engine is not always the right choice for a Semantic Web system which requires scalable performance on special kinds of queries.

There are many applications using inverted indices on Semantic Web data. Many of them are Semantic Web search engines. E.g. Sindice [14] and Watson [6] are used to locate Semantic Web documents, while other search engines such as Falcons [5], SIREn [7], and SemSearch [9] are used for locating Semantic entities, and thus whether to index labels, URLs, literal values or other metadata might differ between them. Occasionally, question answering systems [15, 16] use inverted indices to help identify entities from natural language inputs, which in some sense is also an entity search engine. Despite the categorization, all the above systems index with keywords because the intended usage is to locate relevant resources based on natural language queries posed by users. Our system is very different because the "terms" in our index are no longer keywords but ontological tags. As a result, our index is compatible with entailments sanctioned by the ontologies in the data. This is also why we propose our preprocessing steps prior to indexing, which we have not seen in other works.

8 Conclusion

The contextual tag cloud system is a novel tool that helps both casual users and data providers explore the BTC 2012 dataset: by treating classes and properties as tags, we can visualize patterns of co-occurrence and get summaries of the instance data. From the common patterns users can better understand the distribution of data in the KB; and from the rare co-occurrences users can either find interesting special facts or errors in the data.

In this paper we discuss the underlying computation problem for the contextual tag cloud system. The main problem we solve is to efficiently compute the conditional distribution of types with respect to the intersection of any number of other types. We use an inverted index for this specific kind of query and propose a scalable preprocessing approach. We also propose pruning approaches to save unnecessary queries. We develop formulas to calculate inference under different entailment regimes. Our experiments verify the scalability of both pre and online computation as well as the effectiveness of our pruning approach.

Although the infrastructure described in this paper is specialized for the contextual tag cloud, we believe this infrastructure can be generally applied to other applications. For example, currently we present the visual patterns to users and rely on human intelligence to recognize any common pattern or unlikely co-occurrence. In the future, we will investigate automated algorithms to learn association rules from or detect anomalies in the dataset.

Acknowledgment

This project was partially sponsored by the U.S. Army Research Office (W911NF-11-C-0215). The content of the information does not necessarily reflect the position or the policy of the government, and no official endorsement should be inferred.

References

1. Abadi, D.J., Marcus, A., Madden, S.R., Hollenbach, K.: Scalable Semantic Web data management using vertical partitioning. In: VLDB. pp. 411–422 (2007)
2. Abadi, D., Marcus, A., Madden, S., Hollenbach, K.: SW-Store: a vertically partitioned DBMS for Semantic Web data management. The VLDB Journal 18(2), 385–406 (2009)
3. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.: DBpedia: a nucleus for a web of open data. In: ISWC/ASWC. pp. 722–735 (2007)
4. Broekstra, J., Kampman, A., Van Harmelen, F.: Sesame: A generic architecture for storing and querying RDF and RDF schema. In: ISWC. pp. 54–68 (2002)
5. Cheng, G., Ge, W., Qu, Y.: Falcons: searching and browsing entities on the Semantic Web. In: WWW. pp. 1101–1102 (2008)
6. d'Aquin, M., Motta, E.: Watson, more than a Semantic Web search engine. Semantic Web Journal 2(1), 55–63 (Jan 2011)
7. Delbru, R., Campinas, S., Tummarello, G.: Searching web data: an entity retrieval and high-performance indexing model. Journal of Web Semantics 10(0) (2012)
8. Guo, Y., Pan, Z., Heflin, J.: LUBM: A benchmark for OWL knowledge base systems. Journal of Web Semantics 3(2), 158–182 (2005)
9. Lei, Y., Uren, V., Motta, E.: Semsearch: A search engine for the Semantic Web. Managing Knowledge in a World of Networks pp. 238–245 (2006)
10. Neumann, T., Weikum, G.: The RDF-3X engine for scalable management of RDF data. The VLDB Journal 19(1), 91–113 (2010)
11. Pan, Z., Heflin, J.: DLDB: Extending relational databases to support Semantic Web queries. In: Workshop on Practical and Scaleable Semantic Web Systems. pp. 109–113 (2003)
12. Rohloff, K., Dean, M., Emmons, I., Ryder, D., Sumner, J.: An evaluation of triple-store technologies for large data stores. In: On the Move to Meaningful Internet Systems Workshop. pp. 1105–1114. Springer (2007)
13. Sakr, S., Al-Naymat, G.: Relational processing of RDF queries: a survey. ACM SIGMOD Record 38(4), 23–28 (Jun 2010)
14. Tummarello, G., Delbru, R., Oren, E.: Sindice.com: Weaving the open linked data. In: ISWC/ASWC, pp. 552–565 (2007)
15. Unger, C., Bühmann, L., Lehmann, J., Ngonga Ngomo, A.C., Gerber, D., Cimiano, P.: Template-based question answering over RDF data. In: WWW. pp. 639–648 (2012)
16. Walter, S., Unger, C., Cimiano, P., Bär, D.: Evaluation of a layered approach to question answering over linked data. In: ISWC, pp. 362–374 (2012)
17. Zhang, X., Heflin, J.: Using tag clouds to quickly discover patterns in linked data sets. In: Workshop on Consuming Linked Data (2011)
18. Zhang, X., Song, D., Priya, S., Heflin, J.: Infrastructure for efficient exploration of large scale linked data via contextual tag clouds. Tech. Rep. LU-CSE-13-002, Department of Computer Science and Engineering, Lehigh University