# Exploring the Linked Data Cloud via Contextual Tag Cloud*

Xingjian Zhang, Dezhao Song, Sambhawa Priya,
Zachary Daniels, Kelly Reynolds, and Jeff Heflin

Department of Computer Science and Engineering, Lehigh University
19 Memorial Drive West, Bethlehem, PA 18015, USA
{xiz307,des308,sps210,zad309,ker212,heflin}@cse.lehigh.edu

**Abstract.** We present the contextual tag cloud system, where the context defines a subset of instances, the tags are ontological terms (classes and properties), and the font sizes reflect the number of instances that use each tag. With our system, users can get familiar with the terms and understand how the dataset is populated; or they can dynamically add tags as context and investigate features or look at instances within the constrained subset. With a domain knowledge base, this system helps reveal the patterns of data. For the massive Linked Open Data cloud, it reveals the extent to which data are linked with regard to both the equivalent instances and ontology alignment. In order to achieve this function, we precompute the owl:sameAs closure, and support multiple levels of RDFS entailments. By using an inverted index and pruning algorithms, we design and implement a real time response system for the Billion Triple Challenge dataset.

## 1 Introduction to the Contextual Tag Cloud System

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 [3]? Can we help data providers find potential errors or missing links in a multi-source dataset of mixed quality? We present the contextual tag cloud system as an attempt to address these questions. 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 owl:sameAs links between instances. Thus there are two straightforward ways for representing datasets: We can show axioms related to each term, however sometimes the axioms are missing or too complex to present in a user-friendly way; We can also examine the instances related to each term, however this approach with the most details is also the most time-consuming, and is difficult for users to get an overview of the dataset.

Instead, we think providing a summary of a "type" of data can be more efficient. Ideally, a type is like a class expression, but should not require one to understand the syntax and semantics of description logic. By simply providing the count of each type, the summary will actually reflect the patterns of co-occurrence of ontological terms in

* This submission is for the Billion Triples track in Semantic Web Challenge. Our system is available at http://gimli.cse.lehigh.edu:8080/btc/

instances. From these patterns, a user can get various information: common patterns help users understand the usages of terms[1] and learn which queries are selective and which queries have sufficient data; the rare patterns can lead users to interesting facts. Meanwhile if we visualize the summary properly, not only can we help users better explore the dataset, but also identify possible errors or missing linkages in the data.

**Table 1.** Comparison between traditional Web 2.0 tag cloud and our contextual tag cloud

| | Web 2.0 | Our System |
|---|---|---|
| What is a tag? | A folksonomy defined by users | An ontological term(class or property) |
| What defines the tag size in the tag cloud? | The count of documents marked by the tag | |
| What is a document marked by a tag? | A web page tagged by users | An instance associated with the ontological term in the dataset |
| What happens when a tag is clicked? | Show a list of documents of the tag | Show another contextual tag cloud with this tag added to context |

Our solution is the contextualized tag cloud system. In Table 1 we compare our idea to traditional Web 2.0 tag cloud systems. An instance is like a web page document, but is "tagged" with formal ontological classes, as opposed to folksonomies. Tags, are then another name for the categories of instances. To extend the expressiveness, we include various ways to assign a tag to an instance: (1) **Class** $C$, if the instance has rdf:type of $C$. (2) **Property** $p$, if the instance appears as the subject in one or more triples involving $p$. (3) **Inverse Property** $p-$, if the instance appears as the object in one or more triples involving $p$. (4) **Negation Tag** $\sim t$, if the tag $t$ is not assigned to the instance by the above steps. Note that the semantics of this tag are based on negation-as-failure, which means that anything that is not inferred in the positive is treated as negative. We argue that this is the correct semantics for a system where what is not said is sometimes as important as what is said. We use $Tags(i)$ to denote the set of tags assigned to instance $i$. Then using $T$, a set of tags, we define a new type, or a subset of instances, in which all the instances should have all the tags in $T$.

We define a contextual tag cloud, given a set of tags $T_0$ as the context, as a list of tags $[t_1, \ldots, t_n]$ with various font sizes $[fs_1, \ldots, fs_n]$ that reflects the instance sizes of types $[T_0 \cup \{t_1\}, \ldots, T_0 \cup \{t_n\}]$. Formally, we can define the process of computing the sizes of instances as a function $count$: $2^{\mathcal{T}} \rightarrow \mathcal{N}$. Semantically, given a tag set $T$, $count(T) = |\{i | T \subseteq Tags(i)\}|$. Note that $count(T \cup \{\sim t\}) = count(T) - count(T \cup \{t\})$. We use log functions on the tag frequencies ($count(T_0 \cup \{t_i\})$) to calculate the font sizes so that the tag cloud shows differences of tags in orders of magnitude.

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 inference rules, which

[1] Which may vary from the ontological semantics of the same terms.

helps users understand the dataset better and help data providers investigate the errors in the dataset. Currently we support four levels: **Lv0** no inference, **Lv1** subsumptions only, **Lv2** domain-range inference only, and **Lv3** both Lv1 and Lv2. Similar to super class/property, we define tag $t_1$ a **super tag** of tag $t_2$ if all instances of $t_2$ are entailed to have tag $t_1$ by inference. Note that if property $p$ has rdfs:domain $C$, then $C$ is a super tag of $p$ by Lv2 or Lv3 inference.

This work extends our initial efforts [5] on building a contextual tag cloud system for DBPedia [2]. The BTC dataset is much more complex: from more than 1.4 billion triples, 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. The major contributions presented in this paper include owl:sameAs closure, negation tags and multi-level inference support[2]. We also provide a sorting option, keyword search, paging of results and streaming of answers from the server.

## 2 Using the Contextual Tag Cloud System

The initial tag cloud has context $T = \emptyset$ or semantically $T$ = {owl:Thing}, and the tags in the cloud reflect the absolute sizes of instances related to each tag. We put classes and properties into two separate views, so that users will not treat a property called "author" (which may have domain Publication) as a class name by mistake. To emphasize that difference, we also add an icon with "C" or "P" in front of each tag to indicate the type of tags. 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. A user can also switch to Instance View to investigate the details of instances of the current type.

A user can also change the inference level. By default the tag cloud is for Lv1, the subsumption inference. Usually we can expect tags to become larger when a higher level is chosen. If our inference level entails that a set of tags are equivalent, we choose a cannoncial tag to group them under. We display a ≡ after the canonical tag to indicate this; clicking it will display the equivalent tags. Also for any tag cloud, we can turn on the negation mode, and then the tag sizes indicate how many instances do not have this tag under the current context and inference level. A negation tag can be also added to the context, which mathematically means the relative complement.

With the BTC dataset, a challenging problem for UI design is how we can show so many tags in the tag cloud. A straightforward idea is to show tag clouds in pages. To help users locate interesting tags in the tag cloud, by default we sort the tags by their local names alphabetically. When the system receives a request (a context and an inference level), it will process tags in the same alphabetic order, and then stream out whatever is available for the requested page. If the user chooses to browse tags alphabetically, then the streaming of results is generally able to stay ahead of the user by pre-fetching results for tags on subsequent pages. Instead of browsing, a user can also search for tags by keywords. We index the local name, rdfs:label and rdfs:comment (if it exists) for each

[2] The previous system only supports subsumption inference.

tag to support such keyword search. The retrieved tags will then also be shown in the tag cloud sorted by their relevance to the keyword with their frequencies under the current context and inference level. In addition, we provide sorting by tag frequency as another option, so that users can easily see the most popular tags under the current context and inference. However, we have to wait until all the frequencies are computed to enable this sort option. For some contexts, it can take a few minutes for the overall computation of thousands of pages of results. We show a progress bar of the computation and the estimated time left; and before that is enabled, users can still browse by alphabetical order or search with keywords.

We believe our system can be used for multiple purposes. Here we shall briefly describe two scenarios how a user can explore the BTC dataset.

**Finding interesting facts.** A casual user tries a keyword search on "Manhattan" and the tag cloud is shown in Figure 1. There are classes of parks, streets, etc. located in Manhattan. However, it also has the class "yago:ManhattanProjectPeople"; the user adds this to the context to explore in more detail. In the resulting tag cloud, the user finds various categories for such people, and then searches again for "scientist". Then surprisingly there is a tag "freebase:computer.computer_scientist". The user is intrigued, because she did not know that any computer scientists were involved in the effort to build the first atomic bomb. By adding that tag and switching to the Instance View, she finally learns that this scientist is John von Neumann.

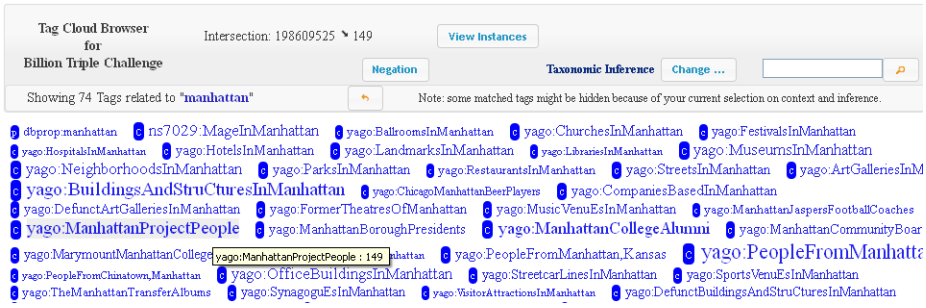


**Fig. 1.** Finding interesting facts: Keyword search with user input "manhattan".

**Examine ontological errors.** Under inference Lv2, a user finds that foaf:Person appears in the tag cloud of context dbpediaowl:Software, implying that some people are software, or vice versa! If the user changes the inference to Lv0 or Lv1, this error will disappear. So that means there must be something wrong with the domain-range inference. If there is a property claimed as having foaf:Person as it domain, then any instance using this property will be classified as the instance of this class. With this assumption in mind, the user adds both foaf:Person and dbpediaowl:Software to the context, selects the property view and inference Lv2, and sorts the properties by frequency. Then the top tag is foaf:homepage, which has all the instances in the current context (by hovering the mouse over the tag, we can see the frequency of this tag). This is very suspicious,

and by clicking on the "P" icon before foaf:homepage, the user can see (in Figure 2) that foaf:Person is an inferred super tag of this tag, and that causes the error[3].

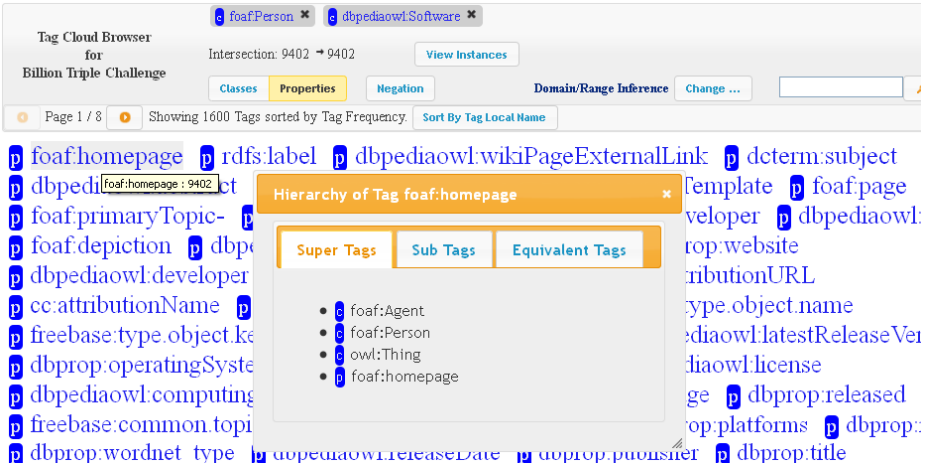


**Fig. 2.** Examining ontological errors. The first property foaf:homepage in the property view implies the foaf:Person class.

## 3 Infrastructure

Our previous experiments [5] show that an RDBMS with decomposed storage mode [1, 4] is not as efficient as using an IR style index for this specific application purpose, both in terms of load time (8X) and online query time (18X). Therefore we continue using the IR approach, but meanwhile add more steps to deal with the BTC dataset.

Our preprocessing is shown in Figure 3, where the dashed rectangles 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 file, and the file of remaining instance triples. The ontology is processed into a closure set of sub-tag axioms for each inference level. We also use the union-find algorithm to compute the closure for owl:sameAs statements, and pick a cannonical id for each owl:sameAs cluster. Then for the instance triples, we replace any instance with its owl:sameAs representative (if exists). 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. Then we use a

[3] Although the domain of foaf:homepage is owl:Thing in the foaf schema, two other sources in the BTC dataset make the claim that the domain is foaf:Person and foaf:Agent respectively.

Unix sort command on this "replaced and flipped" file, so that triples with the same instance subject are clustered together. We use this sorted file together with the multi-level inference to build an inverted index for the instances and tags.

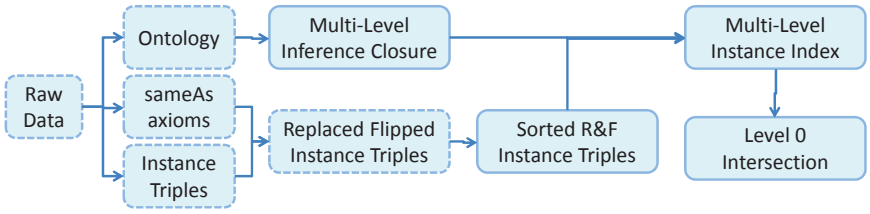


**Fig. 3.** Preprocessing for the tag cloud system

The inverted index is built with tags as indexing terms and each tag has a 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. To support multi-level inference, we have four indexing fields correspondingly. That means we would have four terms for a tag $t$: lv0:t, lv1:t, lv2:t and lv3:t. To trim the index size, we only add an instance to the posting list of the higher level field if the tag is not present in a lower inference level[4], i.e., if an instance $i$ has an explicit tag $t$ in Lv0, then even if $t$ is infered in Lv1, we will not see $i$ in the posting list of lv1:t. Also, since we use negation as failure, we do not need to index negation tags; their size can be calculated from the positive tag and the number of instances matching the context. 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.

After building the index, we go through every tag in lv0, and record all the other tags in lv0 that co-occur with this tag (the co-occurrence of two tags means that there exists an instance having both tags). This precomputation is important because from this we know that some tags will never appear in the tag clouds of some contexts under inference Lv0. We do not compute co-occurece for other levels because that information can be derived from the the lv0 results and the multi-level inference at query time.

Given a context $T$ and an inference level $lv$ as input, the online system will output a list of tag-frequency pairs, where in each pair the tag is assigned to one or more instances of the context $T$ (per inference level $lv$) and the frequency is the number of such instances. With our index, in order to get $count(T \cup \{t\})$ for each tag $t$, we issue an IR query that finds all the instances with all tags in $T$ and $t$. Note that we need to find tags in all the fields corresponding to the inference level. e.g. finding instances with $t$ in Lv3 means an IR clause (lv0:t OR lv1:t OR lv2:t OR lv3:t). So this **count query** is more expensive when the inference level is higher. Since we are issuing a boolean query to an IR index and the result is simply the number of total hits, each individual count

[4] The levels here mean whether the set of rules of one level are contained in that of another. Lv1 and Lv2 are not sorted with respect to each other, i.e., there is a partial ordering of inference levels.

query can be finished in a few milliseconds even with Lv3 (although we must sometime perform hundreds of thousands of such queries).

Figure 4 shows the online query algorithm. First, the input context $T$ will be simplified to $T'$ so that any redudant tags will be removed, e.g., if $T = \{t_1, t_2\}$ and $t_1$ is a super tag of $t_2$ then $T' = \{t_2\}$. This simplified context is used when issuing count queries for candidate tags to appear in the cloud. We use two approaches to try to reduce the candidate set, and thereby the number of count queries needed. We use the precomputed co-occurrence tags in Lv0 and apply our multi-level inference to infer the co-occurrence tags under the current inference level. We also try to get cached results from the previous step. It is very likely that the current request is from a user adding a new tag to the context, thus some results are usually cached. From that result, we extract the tags (ignoring the frequencies); these form a sufficient candidate set of tags because adding a tag to the context will never bring in new candidates. Note that both candidate sets we get are super sets of the tags in the results, and if we can intersect them to get a candidate set that is closer to the true set. In addition, we do not issue count queries on any super tag of the tags in the context since the count will be the same as that of the current context. Also, if we find a tag that does not co-occur with the context, we do not issue count queries on any of its sub tags.

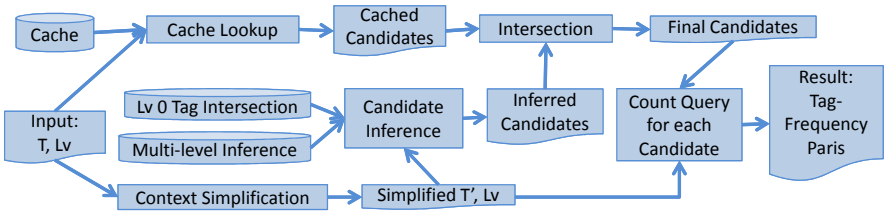


**Fig. 4.** Online computation for the tag cloud system

## 4 Conclusions and Future Work

In order to help both casual users and data providers explore the BTC dataset, we propose our contextual tag cloud system: we treat classes and properties as tags and use contextual tag clouds to visualize the patterns of co-occurrence of ontological terms in the instances specified by the context tags. From the common patterns users can better understand the distribution of data in the KB; and from the rare patterns users can either find interesting special facts or errors in the data.

Our prototype system demonstrates that this novel visualization method can be applied to the BTC dataset. Our future work includes making many improvements, such as optimizing background query performance, improving the user interaction, etc. Also we believe our system can be extended to more specific applications for a focused aspect in our proposed application scenarios, e.g., by allowing users to annotate tag clouds that are interesting or contain errors.

## References

1. Abadi, D.J., Marcus, A., Madden, S.R., Hollenbach, K.: Scalable Semantic Web data management using vertical partitioning. In: 33rd International Conference on Very Large Data Bases(VLDB). pp. 411–422 (2007)
2. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.: DBpedia: A nucleus for a web of open data. In: 6th International Semantic Web Conference and 2nd Asian Semantic Web Conference(ISWC/ASWC). pp. 722–735 (2007)
3. Cyganiak, R., Jentzsch, A.: Linking open data cloud diagram, `http://lod-cloud.net/`
4. 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)
5. Zhang, X., Heflin, J.: Using tag clouds to quickly discover patterns in linked data sets. In: Second International Workshop on Consuming Linked Data (COLD) (2011)

## Appendix

**Minimal requirements**

- *The applications must make use of the Billion Triple Challenge 2012 Dataset.* Yes, all the triples are from the BTC 2012 dataset.

- *The tool or application has to make use of at least the first billion triples from the data provided by the organizers. It is desired that the tool or application uses the complete data set.* Yes, we make use of all the triples: all the instance triples contribute to the tags of instances. Since our system supports RDFS+sameAs inference, some ontological axioms are not utilized.

- *The tool or application is allowed to use other data that can be linked to the Billion Triple Challenge 2012 data set, but there is still an expectation that the primary focus will be on the data provided.* Yes, the only external data source we use is http://prefix.cc that provides useful namespace mapping.

- *The tool or application does not have to be specifically an end-user application, as defined for the Open Track Challenge, but usability is a concern.* Our system is an end-user application for exploring the BTC dataset.

**Additional Desirable Features**

- *The application should do more than simply store/retrieve large numbers of triples.* Yes, our contextual tag cloud system allows users to dynamically select contexts and investigate patterns of instances in the context under a given inference level.

- *The application or tool(s) should be scalable.* Yes, much of our effort focused on achieving a responsive system, and this was done by creating an effective infrastructure and an appropriate UI mechanism.

- *The application or tool(s) should show the use of the very large, mixed quality data set.* Yes, our system provides summary information and supports multiple levels of inference, which is useful even in the presence of errorneous data. Furthermore, our system can help users detect potential errors or missing linkages, which if acted on, can be used to improve data quality.

- *The application should either function in real-time or, if pre-computation is needed, have a real-time realization.* Our online system is responsive and the whole preprocessing takes less than half a day.