Lightweight Semantics over Web Information Systems Content Employing Knowledge Tags

Mária Bieliková, Karol Rástočný

Institute of Informatics and Software Engineering, Faculty of Informatics and Information Technologies, Slovak University of Technology, Ilkovičova 3, Bratislava, Slovakia
{name.surname}@stuba.sk

Abstract. A model of web information system content is crucial for its effective manipulation. We employ knowledge tags – metadata that describe an aspect of an information artifact for purpose of the modeling. Knowledge tags provide a lightweight semantics over the content, which serves web information systems also for sharing knowledge about the content and interconnections between information artifacts. Knowledge tags represent not only content based metadata, but also a collaboration metadata, e.g. aggregations of an implicit user feedback including interactions with the content. To allow this type of metadata we should provide means for knowledge tags repository providing flexible and fast access for effective reasoning. In this paper we address issues related to knowledge tags repository and its automatic maintenance. Main design issues are based on considering dynamic character of the web of information artifacts, which results in content changes in time that can invalidate knowledge tags. We realized the web-scale repository via the MongoDB database. Proposed repository stores knowledge tags in Open Annotation model and supports inference via distributed SPARQL query processing algorithm for MapReduce.

Keywords: lightweight semantics, knowledge tag, annotation, maintenance, MapReduce, SPARQL, distributed repository.

1 Introduction

Effective web content manipulation such as personalized search, recommendation or context aware navigation requires explicit representation of a content model. It obviously includes metadata on the content as an explicit representation of the content model. Moreover, interactions or user activities are recorded and used for intelligent content processing, e.g. employing collaborative filtering techniques. Here, the semantics is often represented implicitly, e.g. by computing similarity of users for relevant information artifact recommendation based on activities of similar users.

For large, dynamic or not completely known information content a lightweight semantics is often the only possible alternative to heavyweight representations, that offer advanced complex reasoning but they cannot be acquired automatically. Lightweight semantics representations form only basic conceptual structures [1]. We propose its representation based on a homogeneous underlying metadata representation:

knowledge tags – metadata that describe an aspect of an information artifact, which can be either content-based or artifact's manipulation-based. Knowledge tag is an extension to the basic concept of a tag as a simple keyword or a term assigned to an information artifact. They represent any metadata that add additional value to the information artifact and represent some knowledge on web information system content. Knowledge tags for web-based information systems can contain explicit and implicit feedback generated by developers working on the source code [2].

Existing web information systems already assign metadata that can be considered as knowledge tags to documents – either manually or automatically [3]. By sharing of these knowledge tags, a new layer of the lightweight semantics over the web information system content can be created. In addition, the knowledge tags layer can become web information systems integration and sharing space based on metadata reuse either on the content itself or characteristics of its manipulation including user interaction or users. Web information systems can take metadata gained by other systems, use it for reasoning a new metadata and share it in the knowledge tags layer again. As a result, web information systems can collaboratively build and improve the semantic layer over the Web content.

However, existing systems obviously store the metadata in their private repositories, so other systems could not use the metadata. Moreover, an issue of dynamic change of information artifacts that can lead to invalidation of knowledge tags should be considered. We present an approach to knowledge tags maintenance, which allows systems to share their metadata in a consistent form via addressing issues related to:

- *The repository:* knowledge tags repository has to store a large amount of knowledge tags in a flexible open format which has to be understandable for web information systems and the repository has to still provide fast parallel access for a numbers of web information systems.
- *Dynamicity of the Web:* a content of the Web repositories is not stable. Stored documents arise, are deleted and modified without a notice. In addition, web users use the web content differently over the time. The Web content instability, diversity in usage of the Web content and also a time aspect can lead to invalidation of knowledge tags (e.g., new and favorite marks) that have to be updated or deleted.

We propose the knowledge tags repository and a method for storing and querying knowledge tags in it. For the repository design it is important to understand requirements for the repository, in particular automatic knowledge tags maintenance. We present also our presumptions on how third-party systems can use knowledge tags.

2 Related Work

Generally, there are two basic issues caused by dynamicity of the Web. Changes in tagged document can have influence to a content of knowledge tags. The influence of modifications in tagged documents to the content of knowledge tags is closely related to a type of knowledge tags and an algorithm, which created metadata stored in knowledge tags. Due to the complexity of change types identification and application to knowledge tags, knowledge tags are often deleted and recreated, although rebuild operations are time expensive and documents modifications require no or only small corrections of knowledge tags in the most of cases.

The second issue is knowledge tags' anchoring in documents, especially in textual documents that are frequently modified documents on the Web. In this case, the knowledge from annotation methods can be utilized, because of knowledge tags and annotations have common characteristics. Both of them are anchored to specific parts of documents and they contain small information on these document parts.

Popular methods of annotations anchoring are based on the start and the end letter indexes of an annotation. But this simple anchoring is not well-usable in dynamic documents on the Web, because web documents are changed without a notice and the index-based anchoring is not automatically repairable without change-set in documents. Moreover, the index-based anchoring is not able to reflex complex modifications, when a document was edited at the annotation's place and the modification has straight influence to both the anchoring and also annotation's content. In this case it is necessary to make decision if the anchoring would be updated or the annotation would become orphaned (new position of the annotation could not be assigned) and also how the annotation's content has to be updated [4].

Phelps and Wilensky proposed a method of robust annotations anchoring [5] with aim to start up development of an anchoring standard. They define criteria of anchoring robustness that are mainly focused on anchoring simplicity and its automatic correction based on new version of anchored document without necessity of a change-set. They also proposed their own anchoring based on SGDOM (simple, general document object model) which describes tree structure of textual documents. Every SGDOM logical part has its own identifier. Phelps and Wilensky utilize these identifiers and define three descriptors of a anchoring – *SGDOM identifier*, *path in a SGDOM tree* and *context*. Each descriptor is tolerant to different document change complexity on the expense of computational complexity.

iAnnotate tool [6] anchors users' annotations in webpages to DOM objects. iAnnotate does not reflect a webpage modifications, but it is focused on changes in the webpage presentation (e.g., a zoom or a change of resolution), to which iAnnotate easily reacts by obtaining new positions of DOM objects. iAnnotate stores annotations in a relational MySQL store which has good performance, but it is not easily distributable and it does not provide necessary flexibility for general annotations.

Anchoring representation based on a tree structure is used in Annotea system [7], too. HTML tree structure is utilized and anchoring descriptors defined by xPath. Annotea repository stores data in RDF model, which gives great flexibility to structure of annotations. Authors did not fully take this great advantage, but this approach to a repository inspired The Open Annotation Collaboration (www.openannotation.org) to a proposition of flexible open annotation model (www.openannotation.org/spec/beta), which allows storing and sharing annotations in unified form.

Annotations in this model consist of:

- *oac:Body*[1] – represents annotation's content which is stored in an annotation;

[1] Namespace oac: OA vocabulary (http://www.openannotation.org/ns/)

- *oac:Target* – represents target document which is annotated. An annotation can contain multiple targets;
- *oac:Constraint* – constrains annotation's body or target to their parts. A constraint on a body (an object of the type *oac:ConstrainedBody* derived from *oac:Body*) is applicable, if only a part of a body is the real body of the annotation. Target constraint (an object of the type *oac:ConstrainedTarget* derived from *oac:Target*) specifies concrete part of the target, to which an annotation is anchored.

Some authors assign annotations only to concrete version of a document and they mark annotations as voided in each other version of a document. In work [8], authors proposed annotations maintenance method based on OA model and Memento framework [9], from which they determine versions of documents, for which annotations were created and after that they filter out voided annotations.

3 Knowledge Tags Maintenance

Current annotation systems support only specific types of annotation for specific types of target documents (e.g., text highlighting in HTML documents). They also provide basic annotations maintenance, mostly in a form of an annotations repository and an automatic anchoring repair based on predefined rules. We are working on knowledge tags maintenance approach to an automatized repair of knowledge tags after updating of tagged documents. A repair of a knowledge tag means discarding of a knowledge tag or updating its anchor and content. If a knowledge tag is not repairable, we mark the knowledge tag as voided and we yield decision how to modify the knowledge tag or if it have to be completely discarded to another system (if it is possible, the system which created the knowledge tag).

We address this goal via the method which consists of three main parts (Fig. 1):

- *Knowledge Tags Repository* – stores knowledge tags in flexible Open Annotation model;
- *Maintenance* – provides automatic maintenance over knowledge tags stored in the knowledge tags repository;
- *Access Provider* – provides access to the repository and notifies Maintenance about updates in the repository and detected new versions of tagged documents.

The maintenance part is responsible for the maintenance of knowledge tags consistency. It means that the maintenance guarantees for a correctness of knowledge tags (their anchoring, content and validity mark) that are provided to web information systems as a reaction to their requests (e.g., loading of knowledge tags anchored to a document). We achieve this via rules for knowledge tags maintenance. These rules are specific to each type of knowledge tags and they are defined by authors of concrete knowledge tag type. We also suppose that the rules are automatically derivable by watching of a knowledge tag life cycle and a life cycle of tagged documents.

The knowledge tags repository is core element, from which usability and performance of whole method is dependent. To achieve overall usability, the knowledge tags repository has to implement flexible and generally acceptable knowledge tags model and provide effective and powerful data access even with non-trivial amount of knowledge tags stored in it.

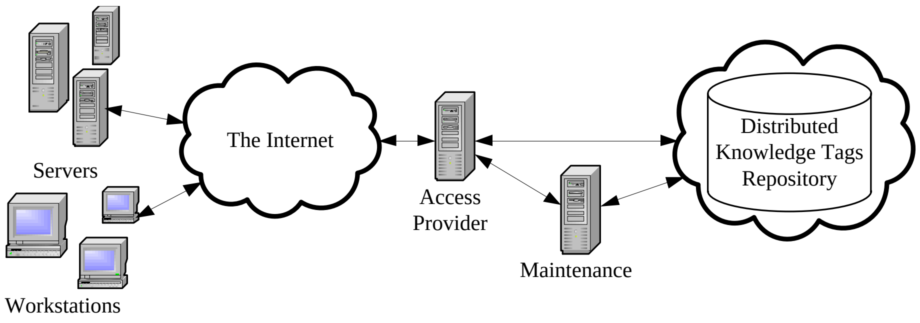

Fig. 1. Architecture of proposed knowledge tags maintenance approach

4 Knowledge Tags Repository

To supply acceptable knowledge tags model, we made decision to utilize existing Open Annotations model, which is currently in beta version but it is already used by a numbers of systems and projects [8], [10], [11]. The model is based on RDF and it is highly recommended to implement it by RDF triple databases and support at least a basic data access (e.g., querying by SPARQL) with RDF serialization output format.

To provide effective and powerful data access, we analyzed standard use cases of annotation repositories and itemize a list of requirements that respect these use cases and specific requirements of OA model and maintenance part of proposed method:

- *Storing of a knowledge tag* – creation of new knowledge tag for a document;
- *Updating of a knowledge tag* – e.g. after modification of a tagged document;
- *Obtaining of concrete knowledge tag* – retrieve the knowledge tag by its URI;
- *Access to knowledge tag's history* – obtaining of previous versions;
- *Obtaining of knowledge tags anchored to a document;*
- *Knowledge tags querying by SPARQL* – compatibility with OA model;
- *Distributed processing* – maintenance over non-trivial amount of knowledge tags.

Manipulation with the whole knowledge tag and not only with its parts is the main component of almost all standard use cases. It is a consequence of the fact that a knowledge tag has sense only as complete information with its content and also with its anchoring in tagged document. But this is in a disagreement with RDF triple data-bases that have good deduction possibilities but, they have serious issue with obtain-ing complete information about an object, when several simple queries have to be processed and each query can take several seconds in large datasets [12]. To address this issue, we set up hypothesis, that we can build efficient RDF-like repository for objects of one type (including types derived from this type) based on another than RDF triple stores, which allows efficient access to complete objects and also supports basic SPARQL query processing with a performance comparable to classical graph-based RDF stores.

4.1 Knowledge Tags Repository Structure

Document databases are in a correlation with our need of an access to whole knowledge tags, while they store documents (in general objects) as one item and not sparse over several tables or collections. This allows fast access to whole objects without necessity of time expensive joins [13]. We decided for MongoDB[2] which matches our requirements: it provides efficient data access (loading and updating) and supports distributed data processing via MapReduce [14].

MongoDB organizes stored objects in collections that allow developers to organize similar or related data to one collection. We design a structure of the knowledge tags repository based on two collections:

- *Tags* – contains knowledge tags in open structure which only have to meet with OA model. The collection provides fast access to knowledge tags by their URI, but access by URI of tagged document is inefficient, because the structure of OA model does not enable effective index over documents URIes;
- *Documents* – contains a mapping of documents to knowledge tags. The mapping has fixed format – a document URI, a knowledge tag URI, a validity of a knowledge tag and access rights. The fixed format allows fast filtrations and access to URIes of knowledge tags anchored to a document.

4.2 Distributed SPARQL Query Processing

MongoDB meets with almost all requirements to the repository. But it does not provide support for SPARQL query processing which is implementable via MapReduce. Several approaches to SPARQL query processing via MapReduce [15], [16] exists already, but all of them are proposed for Apache Hadoop[3] which has some differences in processing of Map and Reduce phases and proposed approaches work with RDF triples stores. MongoDB additionally provides Finalize function for efficient finalization of results of Reduce function.

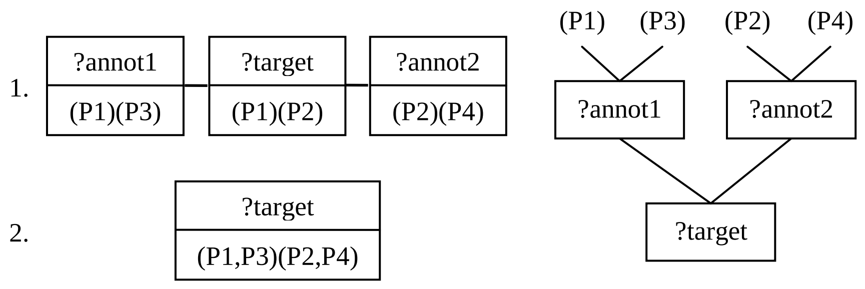

Fig. 2. Iterations of assembly algorithm (join graphs – left, join tree – right) for the example with four triple patterns: P1 – *?annot1 oac:hasTarget ?target*; P2 – *?annot2 oac:hasTarget ?target*; P3 – *?annot1 dcterms:creator ?creator1*; P4 – *?annot2 dcterms:creator ?creator2*

[2] http://www.mongodb.org/
[3] http://hadoop.apache.org/

Our algorithm for distributed SPARQL query processing firstly determines optimal join strategy to minimalize count of necessary join iterations via optimal join tree, the tree with minimal depth. Leafs of the optimal join tree are triple patterns and internal vertexes are join attributes. The tree assembly algorithm runs in a cycle until all join attributes are used in the tree. Each of iterations creates one layer of the tree (Fig. 2):

1. Create a join graph, the graph whose nodes are join attributes with a set of joined triple patterns (if two join attributes have equal set of covered triple patterns, these join attributes are represented by one node) and edges are inserted between nodes with intersecting set of joined triple patterns.
2. Until the join graph is not empty, select one node with the smallest degree and remove the node with incident nodes from the join graph.
3. Add join attributes from selected nodes to the optimal join tree and connect them to vertexes that are in previous layer and join common triple patterns.
4. Join triple pattern from selected nodes to new patterns.

The MapReduce algorithm uses an ordered list of join attributes with their values as a key and a list of deducted objects that consists of an ordered list of joined pattern ids and an ordered list of attributes from patterns with their values as a result. The algorithm is processed in two phases. The first phase is executed with join attributes on the lowest layer of the optimal join tree. In this phase Map function emits results from knowledge tags stored in the repository. Reduce function creates results as Cartesian products of Map results with same keys, where each of newly deducted objects contains complete list of pattern ids, from which it was built and a list of attributes and values from these patterns. Finalize function removes deducted objects that do not have complete list of patterns mapped to processed join keys (see Table 1).

Table 1. Examples of results of Map, Reduce and Finalize functions from the first phase

Function	**Results**

Map	{key : (annot1[X]), value : ({ (P1), (annot1[X] | target[page.html]) }) }
	{key : (annot1[X]), value : ({ (P3), (annot1[X] | creator1[John]) }) }
	{key : (annot1[Y]), value : ({ (P1), (annot1[Y] | target[style.css]) }) }

Reduce	{key : (annot1[X]), value : ({ (P1|P3), (annot1[X] | creator1[John] | target[page.html]) }) }
	{key : (annot1[Y]), value : ({ (P1), (annot1[Y] | target[style.css]) }) }

Finalize	{key : (annot1[X]), value : ({ (P1|P3), (annot1[X] | creator1[John] | target[page.html]) }) }

The second phase iteratively process remaining layers of the optimal join tree. Map function of this phase emits for each result from previous iteration new result with a key from current join attributes and unchanged value. Reduce and Finalize functions are same as in the first phase.

The SPARQL query processing algorithm is optimized for minimal count of join MapReduce cycles (one for each level of the optimal join tree), what decreases a number of necessary time expensive I/O operations between cycles.

5 Evaluation

To evaluate proposed repository, we realized knowledge tags repository solution based on MongoDB and repository based on Bigdata[4] RDF triple database powered by NanoSparqlServer. We selected Bigdata because of its performance and horizontal scalability, what makes possible to store non-trivial amount of knowledge tags. We also looked at in-memory databases (e.g. Trinity or JPregel) that have good performance but they have some issues with horizontal scalability and data persistence. For preliminary evaluation we deploy these repositories only on one node with Intel Core i7-640M processor, 8 GB DDR3 RAM and Seagate Momentus 7200.4 HDD.

During the evaluation we incrementally load one hundred of simple knowledge tags anchored to one (not same) document. Each of knowledge tags consists of sixteen RDF triples in OA model. After each load, we measured times of a load, a retrieving one knowledge tag by its URI, a retrieving URI list of knowledge tags anchored to a document, a retrieving knowledge tags anchored to a document and an execution of simple SPARQL query with one join attribute.

Measured values oscillate around linear function. These oscillations were mainly caused by background system processes and make impossible straight comparison of measured times. For this reason we made linear transposition function of measured values (e.g., Fig. 3) and compared transposed values.

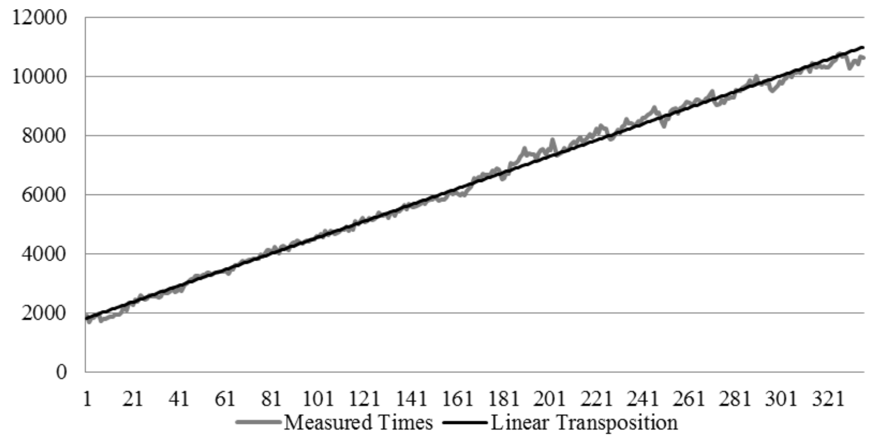

Fig. 3. The dependency of incremental data load duration (in milliseconds) to Bigdata database from a number of knowledge tags (in hundreds) in the repository

The comparison of these two repository realizations shows that the proposed solution based on MongoDB is more effective than the repository based on Bigdata. It is mostly visible on primary operations over knowledge tags. These operations were from 400 to 600 times faster (Fig. 4). Very important is also that less important operation, SPARQL query evaluation, took approximately same time in both realizations.

[4] http://www.systap.com/bigdata.htm

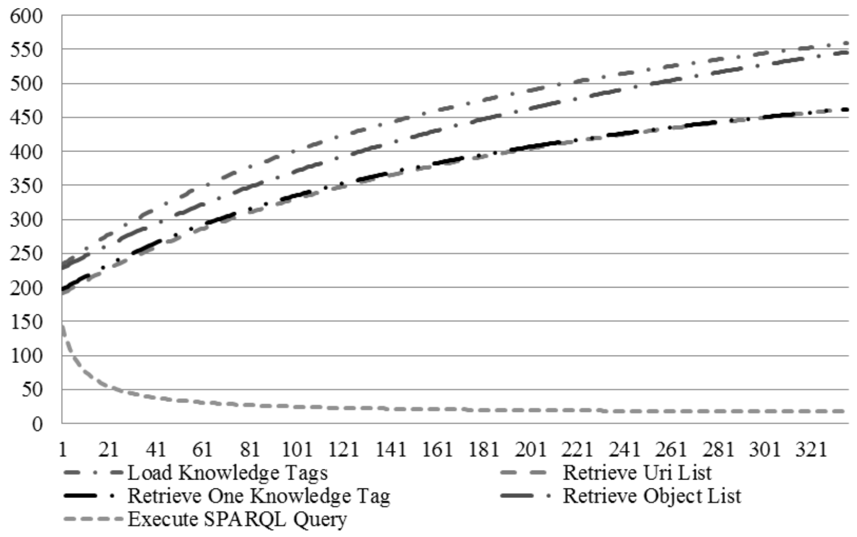

Fig. 4. The comparison of Bigdata and MongoDB realizations. The Y axis presents measured times ratio (how many times is the Bigdata realization slower than the MongoDB realization) and the x axis presents hundreds of knowledge tags stored in the repository in a measuring case.

6 Conclusions and Future Work

In this paper we have presented novel concept of employing knowledge tags for representation of semantics over the web information content. We concentrate on maintaining knowledge tags within proposed knowledge tags repository based on Open Annotation model and realized by MongoDB. The knowledge tags represent lightweight semantics, which includes not only already used content annotations such as relevant domain terms or concepts in educational content [3] or named objects on pictures [17], but also other indices representing various characteristics related to particular information artifact, e.g. its popularity, aggregated users' activity logs or inferred characteristics such as quality. The knowledge tags in that manner create a collaboration space of the web information system, where several web information systems can reuse existing knowledge tags to infer new knowledge tags and so to enrich and improve lightweight semantics over the information space [2].

We also present results of preliminary performance evaluations of the repository. These results indicate that proposed repository is much effective than classical RDF triple stores for our use cases. But for general confirmation of our hypothesis we have to evaluate our approach with more RDF triple databases (including in-memory databases) distributed to several nodes.

Our next steps lead to a proposal of a rule engine based on MapReduce for automatic knowledge tags maintenance. The rule engine employs machine learning to automatically deduce new rules for the maintenance of knowledge tags and improve existing rules by watching of knowledge tags life cycle. These rules are not independent (modification of one knowledge tag can lead to a necessity of modification of several other knowledge tags), so the rule engine should provide effective strategy of rules evaluation.

Acknowledgements. This work was partially supported by the grants VG1/0675/11, VG1/0971/11 and it is the partial result of the Research & Development Operational Programme for project ITMS 26240220039, co-funded by the ERDF.

References

1. Bieliková, M., Barla, M., Šimko, M.: Lightweight Semantics for the “Wild Web”. In: IADIS Int. Conf. WWW/Internet 2011, IADIS Press (2011)
2. Bieliková, M., et al.: Collaborative Programming: The Way to “Better” Software. In: 6th Workshop on Int. and Knowledge Oriented Tech., pp. 89–94. Košice (2011) (in Slovak)
3. Šimko, M.: Automated Acquisition of Domain Model for Adaptive Collaborative Web-Based Learning. Inf. Sciences and Tech., Bulletin of the ACM Slovakia. 2 (4), 9p (2012)
4. Priest, R., Plimmer, B.: RCA: Experiences with an IDE Annotation Tool. In: 6th ACM SIGCHI New Zealand Chapter’s Int. Conf. on Computer-human Interaction Design Centered HCI, 53–60. ACM Press, New York (2006)
5. Phelps, T.A., Wilensky, R.: Robust Intra-Document Locations. Computer Networks. 33, 105-118 (2000)
6. Plimmer, B. et al.: iAnnotate: Exploring Multi-User Ink Annotation in Web Browsers. In: 9th Australasian Conf. on User Interface, pp. 52–60. Australian Comp. Soc., (2010)
7. Kahan, J., Koivunen, M.R.: Annotea: An Open RDF Infrastructure for Shared Web Annotations. In: 10th Int. Conf. on WWW, pp. 623–632. ACM Press, New York (2001)
8. Sanderson, R., Van de Sompel, H.: Making Web Annotations Persistent over Time. In: 10th Annual Joint Conf. on Digit. Lib., pp. 1–10. ACM Press, New York (2010)
9. Van de Sompel, H., Nelson, M.L., Sanderson, R., Balakireva, L.L., Ainsworth, S., Shankar, H.: Memento: Time Travel for the Web. CoRR. abs/0911.1, p. 14 (2009)
10. Gerber, A., Hyland, A., Hunter, J.: A Collaborative Scholarly Annotation System for Dynamic Web Documents: a Literary Case Study. In: 12th int. conf. on Asia-Pacific Digital Libraries, pp. 29-39. Springer-Verlag, Berlin (2010)
11. Yu, C.H., Groza, T., Hunter, J.: High Speed Capture, Retrieval and Rendering of Segment-Based Annotations on 3D Museum Objects. In: Digital Libraries: for Cultural Heritage, Knowledge Dissemination and Future Creation. LNCS, 7008, pp. 5–15. Springer, (2011)
12. Rohloff, K., Dean, M., Emmons, I., Ryder, D., Sumner, J.: An Evaluation of Triple-Store Technologies for Large Data Stores. In: OTM Confederated Int. Conf. on the Move to Meaningful Internet Systems. LNCS 4806, pp. 1105–1114. Springer, (2007)
13. Tiwari, S.: Professional NoSQL. John Wiley & Sons, Inc., Indianapolis (2011)
14. Dean, J., Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters. Communications of the ACM. 51, 107–113 (2008)
15. Kim, H.S., Ravindra, P., Anyanwu, K.: From SPARQL to MapReduce: The Journey Using a Nested TripleGroup Algebra. In: VLDB Endowment 4, pp. 1426–1429. VLDB Endowment, Inc., Seattle (2011)
16. Myung, J., Yeon, J., Lee, S.: SPARQL Basic Graph Pattern Processing with Iterative MapReduce. In: Workshop on Massive Data Analytics on the Cloud, p. 6. ACM Press, New York (2010)
17. Kuric, E.: Automatic Photo Annotation Based on Visual Content Analysis. Information Sciences and Technologies, Bulletin of the ACM Slovakia. 2 (3), 72-75 (2011)