Storage Balancing in P2P Based Distributed RDF Data Stores

Maximiliano Osorio and Carlos Buil-Aranda[*]

Universidad Técnica Federico Santa María, Valparaíso, Chile.
{mosorio,cbuil}@inf.utfsm.cl

Abstract. Centralized RDF repositories have been designed to support RDF data storage and retrieval. However, they suffer from the traditional limitations of centralized approaches which are scalability and fault tolerance, specially in a Web scenario. Peer to Peer (P2P) networks can provide the scalability, fault-tolerance and robustness, features that the current solutions to local RDF storage do not provide. A common strategy from state-of-the-art P2P-RDF data stores is to store triples at three locations so each triple can be found using a look-up by subject, predicate, or object identifier. One major issue of this strategy is the lack of load-balancing, since occurrences in triples are not uniformly distributed. Consequently, this issue leads an unbalanced query processing load distribution and unfair storage load in the network. To solve this problem we propose a new scheme to split the data in the overloaded nodes across neighboring nodes. We propose the use of a Prefix Hash Table consisting in XXX to access to such data. We provide an empirical evaluation of our approach and compare with other state of the art systems for storage balancing showing the feasibility of our approach.

1 Introduction

Semantic Web applications rely on data that is stored in centralized RDF repositories and look-up systems. These systems have been designed to support RDF data storage and access, however, they normally suffer from the traditional limitations of these approaches which are scalability and fault tolerance [1]. These limitations are more evident on a Web environment due to the large amount of clients accessing concurrently the data stored in them. As an alternative to these centralized systems, P2P approaches have been proposed to overcome some of their limitations by building (fully) decentralized data storage and retrieval systems [2].

P2P networks provide the scalability, fault-tolerance and robustness features needed by Internet applications. However, it is crucial for the P2P system to distribute evenly the data over the network, otherwise, the P2P system becomes a centralized system. This problem already exists in some of the state of the art

[*] Supported by the Millennium Nucleus Center for Semantic Web Research under Grant NC120004

RDF-P2P systems like Atlas [3], being a major cornerstone of such networks. In this paper, we propose a structure for indexing triples using Distributed Hash Tables (DHT) and Prefix Hash Tree (PHT). Our approach is based on evenly distributing the excess of data across neighboring nodes based on a dynamic changing threshold. For maintaining fast access to the data stored we propose to use a ring-based P2P system and a distributed data structure called Prefix Hash Tree (PHT). In PHT each node in the trie has a label with a prefix that is defined recursively to allow fast access to the nodes in the network containing the related data. The rest of the paper is organized as follows: in Section 2 we describe the existing state of the art related to RDF storage in P2P networks. Next in Section 3 we present our solution to the problems identified in Section 2. We evaluate our approach in Section 4 and finally we present our conclusions in Section 5.

2 Related work

The Resource Description Framework (RDF) [4] is the recommended data model by the W3C aiming to improve the World Wide Web with machine-processable semantic data for data interchange on the Web. The notion of RDF triple is the basic building block of the RDF model. It consists of a subject (s), a predicate (p) and an object (o). More precisely, given a set of IRI [5] references I, a set of blank nodes B, and a set of literals L, a triple $(s, p, o) \in (I \cup B) \times I \times (I \cup B \cup L)$. The subject of a triple denotes the resource that the statement is about, the predicate denotes a property of the subject and the object presents the value of the property.

Centralized RDF repositories and lookup systems such as Jena[1], RDFDB[2] or Virtuoso[3] have been designed to support RDF data storage and retrieval. Although these systems are highly optimized RDF stores, they suffer from the traditional limitations of centralized approaches such as scalability and fault-tolerance in an open Web scenario [1]. As an alternative to these centralized systems, P2P approaches [6,7,8] have been proposed to overcome some of these limitations by building decentralized data storage and retrieval systems [2].

P2P networks and especially distributed hash tables (DHTs [9], hash tables in which the responsibility for maintaining the mapping from keys to values is distributed among the nodes in the network) have gained much attention recently [2]. The reasons are the scalability, fault-tolerance and robustness features key features for most Internet and Web applications. In such networks a responsible node is the node that stores the key for the RDF triple. Some RDF systems use DHTs to store and query RDF data at Internet scale like RDFPeers [10], Atlas [3], 3rdf [11] or GridVine [12]. We can classify these solutions according to the overlay structure (ring-based, cube-based, tree-based and generic).

[1] http://jena.apache.org/documentation/rdf/

[2] https://github.com/scor/rdfdb

[3] https://virtuoso.openlinksw.com/

RDFPeers [10] and Atlas [3] use a ring-based structure. In DHT, each peer and data item has an identifier, i.e. the network address and the file name, are hashed to a hash key in key space $[0, 2m)$ for a typical constant $m = 128$ for 128-bit keys. A peer then gets all the data assigned which has a hash key between its hash key and the next larger hash key of another peer in the key space ring. In this way the key space range assigned to any peer is not greater than factor O $(logn)$.

GridVine and 3rdf [11] are distributed RDF systems using search-tree based overlay networks, such as the P2P systems P-Grid [13] and 3nut [14]. The difference between a search tree instead a hash table is omitting any hashing of data keys. The objective of this approach is to preserve the order of data key in key space and achieves efficient range queries in key space. The trade-off is in this class of overlays has more complexity, a larger routing structure being more difficult to maintain [15].

In ring-based solutions index triples 3 times for each component of triple, these are regularly disseminated to the nodes in the network by calculating the hash function of the subjects, predicates, and objects and sending the triples to the nodes responsible received. This indexing technique provides the possibility to find triples based on any search criteria as long there exist at least one constant in a triple pattern [16]. The triples distribution frequency is not uniformly distributed, thus, it is possible that the responsible peer will be heavily loaded. In [10] the authors approach the storage balancing problem by simply not indexing the most common RDF triples such those having an `rdf:type` predicate and the requesting node must then find an alternative way of resolving the query and ask to another target node. The result of such an approach is cost of possibly losing the complete result.

To efficiently balance the RDF triples, authors in [17] propose the use of four RDF databases in each peer of the network (local triples, received triples, generated triples, and replica triples database). The local triples database stores RDF triples that originate from the particular node, the received triples database stores all local triples, these are regularly disseminated to the nodes in the network by calculating the hash function of the subjects, predicates, and objects and sending the triples to the nodes responsible received. Also, each node hosts a database for generated triples that originate from forward chaining. Finally, the replica database has a copy of the data of whose IDs that are the nearest to the target hash value determined by the hash function. The authors addressed the issue of load-balancing to build an overlay tree over a DHT, if a node detects that it is overloaded, the node performs a split operation, half of the triples remained in the local part of the remote triples database and half of them were moved to the new node. This approach allows an easier access to the data, however the overhead generated by the five databases in each node of the network and the use of a forward chaining approach for RDF(S) reasoning results in higher storage and bandwidth costs for a single peer [18].

Atlas [8] is a distributed RDF system that uses DHTs for distributing RDF data across the peers. For storing an RDF triple Atlas sends three DHT put requests using as key the subject, property and object together, and the triple itself as an item. The key is hashed to create the identifier that leads to the responsible node where the triple is stored. Atlas also suffers from load imbalances [18], due to RDF triples some triple components are more frequent, for instance `rdf:type` and `rdfs:label`, the peer responsible for such a key store more triples and the built-in load balancing is not able to balance this higher load.

In [16] the authors propose an indexing scheme "3-tuple index" where 3 combined routing indexes are created on triples subject, predicate and object components. The output combinations are subject+predicate, predicate+object and object+subject.

In summary, for storing and accessing RDF triples in a P2P network the main approach is to use DHTs. However one critical problem is not correctly balancing the amount of data stored in each node, leading to extra processing by some nodes. We now proceed to describe our solution to such problem for ring-based solutions, based on Atlas.

3 System model and data model

An obvious approach to solve the node saturation problem described before is to add a threshold limit indicating the maximum amount of triples that a node can store. Once a node reaches that threshold, the node splits the data into two children nodes, redistributed the data keys among the children equally like in P-Grid [13] (without using most of the P-Grid features like the routing layer). After that, each child stores half of the triples that can also be split again in case that node reaches again the threshold limit. Now the problem is to find the triples in the tree since a triple can be placed in any node of the tree. To fix such problem, we propose to use a distributed data structure called Prefix Hash Tree (PHT). In PHT each node in the trie has a label with a prefix that is defined recursively to allow fast access to the nodes in the network containing the related data. Consider the node l in which l_0 and l_1 are the left hand and right hand side nodes. For each two children we have the following properties:

1. Each node has either 0 or 2 children.
2. A key K is stored at a leaf node whose label is a prefix K.
3. Each leaf node stores at most B keys.
4. Each internal node contains at least $(B + 1)$ keys in its sub-tree.

The PHT structure is a routine binary trie where each node in the trie is a node of the DHT ring. Atlas uses a mapping dictionary, which maps a unique integer value to a triple. Since URIs and literals may consist of long strings, these are mapped to integer values and then mapped again to the triple storage. Finally, the query evaluation is performed using these integer values [8]. Using the previous idea, the domain for the indexing is $\{0, 1\}^D$, thus, the triples are stored according to their integer value. Figure 1 illustrates two cases, in the first case (artist,o,p) the node is able to store the triples while in the second case

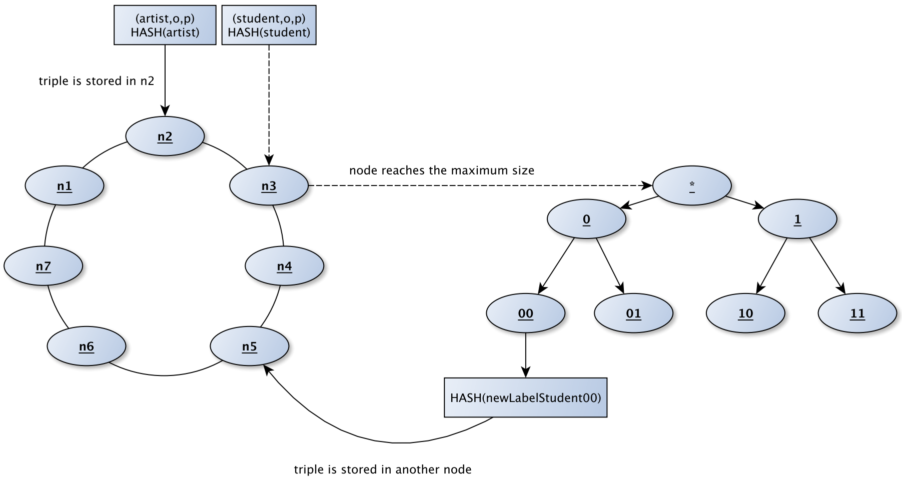


Fig. 1: Triple (artist,o,p) is stored in n2, in this case the node n2 can store the triple. Triple (student,o,p) has to be stored in n3, n3 cannot store the triple, then n3 use the PHT and the triple is stored in n6.

(student,o,p) the node reaches the maximum size and use the prefix Hash Tree (PHT).

Notice that this problem cannot be solved by just redistributing again the triples across the network since the hash function for distributing the data would return similar keys all the time and some nodes would be still overloaded. Besides generating more complexity at search time.

3.1 Operations

Lookup Given a part of the triple K, PHT lookup returns a unique leaf node: $leaf(K)$. To find the triple our implementation uses a binary search algorithm: if the current prefix is an internal node, the search tries a shorter prefix, and if the current prefix is not an internal node, the search tries a longer prefix.

Query Forwarding queries within the tree does not consume expensive DHT routing and can be done via direct communication. A query reaches its destination in $O(log(N) + d)$ steps, where N is the number of nodes in the DHT network and d is the depth of tree.

Insert/Delete Insertion is a common operation in the system. When a new triple has to be inserted, the node calculates the hash function for subject, predicate, and object and sends the triples to the responsible nodes. When a node receives the triple for either insertion or deletion, the node verifies whether the threshold limit B is reached or not. If not, the node stores the triple, otherwise, the node performs `lookup` to find the new responsible node and saves the triple in it if the triples do not reach the Maximum value amount of triples B . That integer value of the mapping dictionary is saved in an ATLAS local database. Similarly, deletion can cause that a subtree to collapse into a single leaf node and then the corresponding balancing operations are executed.

4 Performance Analysis

We compared our solution for storage balancing with the Atlas algorithm and with the approach in [16]. To do that we simulated a P2P with 1,000 nodes and distributed among these nodes 1,000,000 triples from DBpedia, the frequency per predicate is shown in Table 1. We evaluated how well the data is distributed across these 1,000 nodes, and we mark as future work to evaluate the query execution of a complete SPARQL benchmark. The source code for our evaluation and the data used can be found in `http://inf.utfsm.cl/~mosorio/p2p_rdf_balance`.

Frequency	subject/object/predicate
1000000	rdf:type
7945	Category:Living_people
1868	Category:Articles_containing_video_clips
1805	Category:Townships_in_Minnesota
1343	Category:American_films
1246	Category:Towns_in_Wisconsin
1212	Category:English-language_films
1045	Category:Townships_in_Michigan
944	Category:Townships_in_Pennsylvania
928	Category:Cities_in_Iowa
917	Category:Villages_in_Illinois
915	Category:Cities_in_Texas
890	Category:Towns_in_New_York
842	Category:Cities_in_Minnesota

Table 1: Frequency per predicate

Storage balancing We analyze the effect of data indexing algorithms from Atlas and [16]. The goal is evaluate the distribution's quality of these solutions. In the figure 2 we present the results for each algorithms indexing 1 million triples from DBpedia, Figures 2a,2b show the results for Atlas's algorithm. In Figure 2a there is an outlier point which corresponds to the predicate `rdf:type`. The figure 2b has the same results but we remove the node responsible for `rdf:type` but we can see multiple outliers.

On the other hand, [16] proposes a mixed approach, 2c shows the results, in comparison with Atlas's results we see an improvement in the distribution but we see multiples outliers. On the other hand, the figure 2d shows the results for the proposed algorithms using the maximum number of triples by node $B = 1000$, in comparison with the last two algorithms, we can see that the triples are well distributed and there are not outliers in the results.

In our solution the B parameter determines when the data redistribution should happen (which can be adjusted automatically). We thus evaluate the algorithm with different B configurations. Figures 2d, 2e, 2f show the results using different amount of triples and a different values for B.

In our approach the tree's depth may vary using different values for B. Table 2 show the different tree depths depending on the value of the B parameter, being all of them low. Moreover, the depth of tree decreases when the B increases. If the number of triples into the node increases over time, the parameter B can be adjusted dynamically becoming an adaptive process.

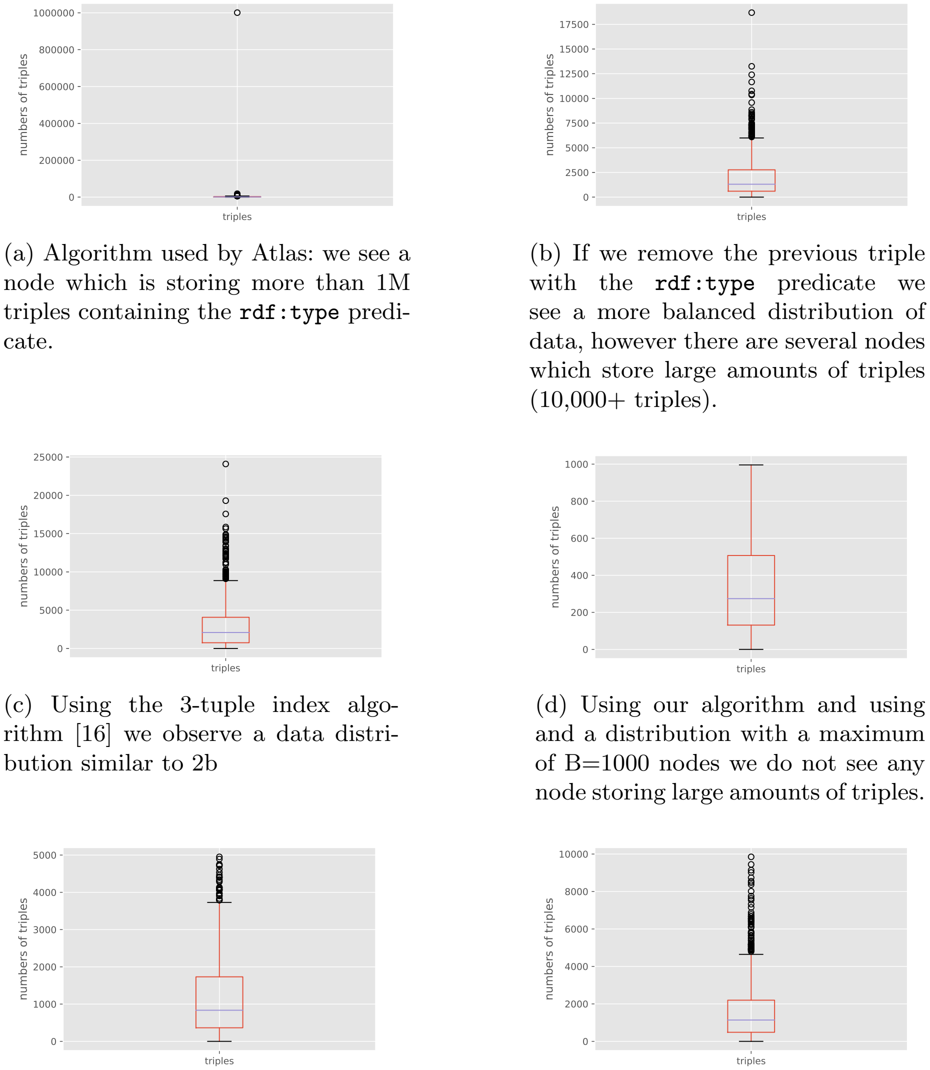
	Number of triples		
B	100.000 triples	1 million triples	10 million triples
1000	2	6	10
5000	2	2	6
10000	1	2	5

Table 2: Tree's depth using different numbers of triples and B

5 Conclusions

In this paper we discussed indexing techniques for RDF in P2P networks. We identified a problem of storage balancing in Distributed Hash Tables (DHT) for RDF stores and we proposed a solution to it by using a new data structure called Prefix Hash tables (PHT). Using PHT we are able to achieve a better storage distribution that the existing techniques in literature showing it empirically. We evaluated PHT using different configurations demonstrating that PHT can achieve a fair stability when distributing the RDF data across the P2P network. Finally, we showed that the tree's depth is fair and considering that the worst case message complexity of this solution is $O(log(N) + D)$ we can conclude that new scheme does not high;y increase time complexity.

In summary, the proposed algorithm limits the maximum number of triples by each node, if the node reaches the maximum number the node performs a split operation. Also, using PHT, a node can query and find the triple in the tree. The time overhead by using PHT and the maximum number is low: $O(log(N) + D)$.



(a) Algorithm used by Atlas: we see a node which is storing more than 1M triples containing the `rdf:type` predicate.

(b) If we remove the previous triple with the `rdf:type` predicate we see a more balanced distribution of data, however there are several nodes which store large amounts of triples (10,000+ triples).

(c) Using the 3-tuple index algorithm [16] we observe a data distribution similar to 2b

(d) Using our algorithm and using and a distribution with a maximum of B=1000 nodes we do not see any node storing large amounts of triples.

(e) Using our algorithm and using and a distribution with a maximum of B=5000 nodes we see a few node storing medium amounts of triples.

(f) Using our algorithm and using and a distribution with a maximum of B=10000 nodes we observe a data distribution similar to 2e.

Fig. 2: Indexing 1 million triples from DBpedia

6 References

1. C. Buil-Aranda, A. Hogan, J. Umbrich, and P.-Y. Vandenbussche, "SPARQL Web-Querying Infrastructure: Ready for Action?," in *ISWC2013*, pp. 277–293, 2013.
2. I. Filali, F. Bongiovanni, F. Huet, and F. Baude, "A survey of structured p2p systems for rdf data storage and retrieval," in *Transactions on large-scale data-and knowledge-centered systems iii*, pp. 20–55, Springer, 2011.
3. Z. Kaoudi, M. Koubarakis, K. Kyzirakos, I. Miliaraki, M. Magiridou, and A. Papadakis-Pesaresi, "Atlas: Storing, updating and querying rdf (s) data on top of dhts," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 8, no. 4, pp. 271–277, 2010.
4. "Rdf - semantic web standards." `https://www.w3.org/RDF/`. (Accessed on 06/20/2017).
5. M. Dürst and M. Suignard, "Internationalized resource identifiers (iris)," tech. rep., 2004.
6. W. Nejdl, B. Wolf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, M. Palmér, and T. Risch, "Edutella: a p2p networking infrastructure based on rdf," in *Proceedings of the 11th international conference on World Wide Web*, pp. 604–615, ACM, 2002.
7. E. Della Valle, A. Turati, and A. Ghioni, "Page: A distributed infrastructure for fostering rdf-based interoperability," in *DAIS*, vol. 6, pp. 347–353, Springer, 2006.
8. Z. Kaoudi, I. Miliaraki, and M. Koubarakis, "RDFS reasoning and query answering on top of DHTs," *The Semantic Web-ISWC 2008*, pp. 499–516, 2008.
9. H. Balakrishnan, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Looking up data in p2p systems," *Communications of the ACM*, vol. 46, no. 2, pp. 43–48, 2003.
10. M. Cai and M. Frank, "Rdfpeers: a scalable distributed rdf repository based on a structured peer-to-peer network," in *Proceedings of the 13th international conference on World Wide Web*, pp. 650–657, ACM, 2004.
11. L. Ali, T. Janson, and G. Lausen, "3rdf: Storing and querying rdf data on top of the 3nuts overlay network," in *Database and Expert Systems Applications (DEXA), 2011 22nd International Workshop on*, pp. 257–261, IEEE, 2011.
12. K. Aberer, P. Cudré-Mauroux, M. Hauswirth, and T. Van Pelt, *GridVine: Building Internet-Scale Semantic Overlay Networks*, pp. 107–121. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004.
13. K. Aberer, P. Cudré-Mauroux, A. Datta, Z. Despotovic, M. Hauswirth, M. Punceva, and R. Schmidt, "P-grid: a self-organizing structured p2p system," *ACM SIGMOD Record*, vol. 32, no. 3, pp. 29–33, 2003.
14. T. Janson, P. Mahlmann, and C. Schindelhauer, "3nuts: A locality-aware peer-to-peer network combining random networks, search trees, and dhts," 2009.
15. L. Ali, T. Janson, G. Lausen, and C. Schindelhauer, "Effects of network structure improvement on distributed rdf querying," in *International Conference on Data Management in Cloud, Grid and P2P Systems*, pp. 63–74, Springer, 2013.
16. L. Ali, T. Janson, and C. Schindelhauer, "Towards load balancing and parallelizing of rdf query processing in p2p based distributed rdf data stores," in *Parallel, Distributed and Network-Based Processing (PDP), 2014 22nd Euromicro International Conference on*, pp. 307–311, IEEE, 2014.
17. D. Battré, F. Heine, A. Hoing, and O. Kao, "Load-balancing in p2p based rdf stores," in *2nd Workshop on Scalable Semantic Web Knowledge Base System*, 2006.
18. Z. Kaoudi, *Distributed RDF query processing and reasoning in peer-to-peer networks.* PhD thesis, 2011.