CLUSTER MECHANISMS IN A SELF-ORGANIZING DISTRIBUTED SEMANTIC STORE

Marko Harasic, Anne Augustin, Robert Tolksdorf and Philipp Obermeier
AG Networked Information Systems, Freie Universität Berlin, Königin-Luise-Str. 24/26, D-14195 Berlin, Germany

Keywords: Self-organization, Semantic web, Clustering.

Abstract: Distributed semantic stores can employ self-organization principles to improve their scalability. We present an implementation that uses ant colony optimization for clustering similar semantic information facilitating scalable retrieval. For the clustering mechanisms we use similarity measures that do not rely on access to a complete ontology. We describe a syntactical and a fingerprint-based similarity measure and discuss them regarding to expressiveness and computational effort. The results of an evaluation show that, with increasing volume of data and number of processes, the fingerprint-based measure performs much better than the syntactical one. We conclude with a discussion how to combine the advantages of the two measures and propose some technical enhancements improving the efficiency of the system.

1 INTRODUCTION

In adherence to the Semantic Web's distributed nature, RDF knowledge bases are, in general, scattered across multiple server machines on the Internet. However, there are still no coherent solutions for efficient distributed storage and retrieval for these vastly growing amount of data. Admittedly, there are a number of very powerful RDF triple stores handling millions or even billions of triples, e.g. Virtuoso Server[1], BigOWLIM[2] and Allegro Graph[3]. These are designed, though, to operate on a central server machine or a computer grid comprising a limited number of hosts still depending on centralized components. Therefore, these concepts cannot scale for the millions of web servers as they exist on the Internet today. To cope with such huge server networks, our approach is to use self organization mechanisms like they are, for example, found in ant colonies to decentralize the storage access, preventing bottlenecks and as a result provide a scalable highly distributed store for semantic information. In this paper we present a distributed RDF store where triple storage and retrieval is performed by simple virtual ants which cluster similar triples in a server network. The resulting system gets along without any central control, triple writing and retrieval can be executed from each node in the network enabling load-balanced requests.

The rest of this paper is organized as follows. In Section 2 we give a short introduction in RDF and SwarmLinda, present two other approaches of distributing RDF stores. We introduce in section 3 the underlaying algorithms for writing an retrieving data, while in section 4 we present the similarity measures used by the algorithms of section 3. Section 5 gives a brief overview about the implementation. Finally sections 6 and 7 show and discuss the test results of this paper and gives an outlook over future work.

2 RELATED WORK

SwarmLinda (Menezes and Tolksdorf, 2003) is a Linda-System (Gelernter, 1985) that uses swarm algorithms to store and retrieve data from a server network. Data is stored as tuples that contain several typed fields and retrieval is performed on the base of templates that match these tuples. For example to retrieve a tuple <"zebrafish",55> from the store a template <String, int> could be used as well as <"zebrafish",int> or <String,55>. In analogy to ants in nature that perform foraging and brood sorting tasks (Bonabeau et al., 1999), tuples in SwarmLinda that

[1] http://virtuoso.openlinksw.com/
[2] http://www.ontotext.com/owlim/big/
[3] http://www.franz.com/agraph/allegrograph/

Harasic M., Augustin A., Tolksdorf R. and Obermeier P.
CLUSTER MECHANISMS IN A SELF-ORGANIZING DISTRIBUTED SEMANTIC STORE.
DOI: 10.5220/0002808401310136
In *Proceedings of the 6th International Conference on Web Information Systems and Technology (WEBIST 2010)*, page
ISBN: 978-989-674-025-2

match the same template are clustered and trails are left in the system to make these clusters traceable.

In (Tolksdorf and Augustin, 2009) and (Koske, 2009) the ant colony algorithms of SwarmLinda were adopted to realize a distributed storage for RDF triples in which similar triples are clustered. RDF (Resource Description Language)[4] is a language to represent information about resources on the Web in a machine readable way. An RDF triple is used to describe a resource that is identified by a URI. An RDF triple always has the form (S,P,O), where S is the subject and P and O the predicate and object of the triple. For example, with the triple(*http://birds.org/description/onto.rdf#sparrow*, *http://animals.org/livesOn*, *http://plants.org/grains*) we can state that sparrows live on grains.

To overcome the scalability issues for storing millions of triples, P2P-algorithms where adopted to fit the needs of RDF-storage. While Edutella (Nejdl et al., 2001) bases on a Gnutella like approach, RDFPeers (Cai and Frank, 2004) uses a distributed hash table (DHT) to compute the storage locations of RDF-triples. Edutella offers a simple and cost effective way to create the P2P-network for an arbitrary large number of peers, query processing suffers from a large number of peers since queries are processed by flooding the network creating a large overhead of traffic. On the other side the DHT approach of RDFPeers guaranties the routing of queries in $O(\log n)$ messages, but DHTs are costly to maintain. In our approch we try to combine the benefits of both approaches without their drawbacks. The network is created as simple as possible like Edutella and the routing with pheromones makes it possible to answer a query as fast as RDFPeers by using the optimal path to the location of the triple. Since the dimension d of power law graphs is usually limited to $O(\log n)$, a query can be processed with only $O(\log n)$ messages. In this work we present an implementation of a triple storage using optimised versions of the algorithms from (Tolksdorf and Augustin, 2009) and provide an evaluation of this system in respect of scalability comparing the syntactical similarity measure from (Tolksdorf and Augustin, 2009) and a fingerprint based similarity measure.

3 ALGORITHMS

To store and retrieve RDF triples from the system the `write` and `read` procedures are used which are performed by `w`- and `r`-ants. Ants communicate indirectly by depositing information on the nodes. These so-called pheromones evaporate over time, making the system adaptive to changes in network topology and stored content. All ants behave autonomously and probabilistically and use only local information to compute their decisions. Each node holds a routing table, maintained by the passing ants and which they use for returning to the node of request.

3.1 Triple Writing

In the system, three copies for each inserted triple exits, which each is clustered in regard to the triple's subject, predicate and object. As a result each node holds three types of clusters, a subject, predicate and object cluster which contain the triples that were clustered regarding their subjects, predicates and objects, respectively. Thus for a triple to be stored three `w`-ants are generated which roam the network to find a suitable cluster for one of the triple's resources respectively. In the following we will refer to this resource as *cluster resource*. The behavior of a single `w`-ant is as follows. The `w`-ant starts living at the node of the request and carries one triple copy and a resource pointer $r \in \{S,P,O\}$ which indicates the triple's cluster resource. In order to decide if the triple should be dropped on the current node the ant measures the similarity of its cluster resource r to the cluster resources $r_1 \dots r_n$ in the appropriate resource cluster. The normalized sum of the resulting similarity values

$$Sim = \frac{1}{n}\sum_{i=1}^{n} sim(r, r_i) \tag{1}$$

is used for computing the drop probability.

$$P_{drop} = \left(\frac{Sim}{Sim + c_{drop}}\right)^2 \tag{2}$$

In equation 2 *Sim* is an exchangeable similarity measure to determine the similarity between two resources and c_{drop} a constant value which modifies the likelihood of dropping the triple on the current node.

Based on P_{drop} the `w`-ant decides whether to drop its triple on the current location. If it does it stores it in the appropriate cluster and walks back to its origin to report the success of the writing procedure. On its way back it lays its cluster resource as pheromone on each edge it takes. While the ant does not decide to drop the triple it roams the network selecting paths which are marked with pheromones of its cluster resource. The probability to change from the current node i to a node j in its neighborhood $NH(i)$ is given by the following equation.

$$P_{ij} = \frac{Ph_j(cr)}{\sum_{n \in NH(i)} Ph_n(cr)} \tag{3}$$

[4] http://www.w3.org/RDF/

In equation 3 $Ph_k(cr)$ is the amount of pheromones matching the cluster resource cr on node k. The `w`-ants age with each node change preventing them from wandering around in the network endlessly. When a `w`-ant reaches a certain age it drops its triple on the current location whether it fits there or not and returns back to its origin, spreading the pheromone $Ph_k(cr)$ corresponding to its cluster resource cr along the path traveled.

3.2 Triple Retrieval

Triple retrieval is performed by `r`-ants which return all found triples matching a given pattern. For example, to retrieve the triple (S,P,O) from the system we can invoke `read<(S,?P,?O)>`. The pattern matches all triples that have S as subject. For each operation, one `r`-ant is generated which looks for a triple that matches the pattern. For example, if the pattern is $(S,?P,?O)$ the ant follows S-pheromones $Ph(S)$ to locate the appropriate cluster using equation 3 for the path selection. `r`-ants which find a matching triple or a set of matching triples return to the node of the request and like the `w`-ants leave pheromones on their way back. The `r`-ants also age with each hop in the network, meaning that if they are not successful in finding a matching triple after having visited a certain number of nodes, they return to the node of the request without result, not leaving a pheromone trail. Because of the pure probabilistic behavior of the whole system, an empty result does not imply the absence of matching triples, leaving the client the decision to rerun the reading process. Following operations for the same query use `w`-ants with increased maximum age, realizing a deeper search.

4 MEASURES

The algorithms above work with exchangeable similarity measures that lead to different kinds of clusters. In the following we will present two different measures. The syntactical similarity measure leads to clusters of triples that resemble in their contained URIs but has more computing complexity. The vector similarity measure does abstract entirely from the actual similarity of the resources minimizing the computational costs of the ants.

4.1 Clustering based on URI Similarity

The first similarity measure that we used to cluster similar triples was taken from (Tolksdorf and Augustin, 2009) and compares the namespace of two URIs. The similarity of two hierarchical URIs is measured by comparing the host and path component separately. The host and path themselves are splitted into their hierarchical components which are compared pairwise by applying the Levenshtein-distance *edit*. The comparison of the segmented paths $m = m_1/m_2/\ldots/m_k$ and $n = n_1/n_2\ldots/n_l$ of two URIs is given as follows.

$$sim_{path}(m,n) = \sum_{i=1}^{min(k,l)} c_i edit(m_i,n_i) \qquad (4)$$

with

$$c_i = \frac{2^{max(k,l)-i}}{2^{max(k,l)}-1}$$

as a weighting function and *edit* as the normalized Levenshtein-distance of two strings. As a result of the weighting function, a path segment a level higher in the hierarchy is weighted double. To compare the URIs *http://birds.org/description/onto.rdf#sparrow* and *http://birds.org/description/onto.rdf#duck*, first the hosts would be compared which are equal in this case and then the paths would be compared. The paths would be splitted into three parts, the first one into *description*, *onto.rdf* and *sparrow* and the other one into *description*, *onto.rdf* and *duck*. The similarity between the two paths is then ≈ 0.86. The similarity of two hosts is computed comparing their domain-labels pairwise, starting with the hierarchical highest label which is weighted highest. The weighted sum of the host- and path comparison values results in the overall similarity of the regarded URIs u_1 and u_2.

$$\begin{aligned} sim_{URI}(u_1,u_2) &= c_1 \cdot sim_{host}(u_1,u_2) \\ &\quad + c_2 \cdot sim_{path}(u_1,u_2) \end{aligned}$$

To achieve that the paths only differentiate the URIs if the hosts are equal (or very similar) c_1 is set to 0.9 and c_2 to 0.1, so that the host is weighted ninefold. So the overall similarity for the URIs above would be $0.9 \cdot 1 + 0.1 \cdot 0.86 = 0.986$. An overview of the comparison is shown in figure 1.

In order to compute *Sim*, an ant has to compare r with all cluster resources on the current node. Since computing the Levenshtein-distance is rather costly, this measure suffers from a growing amount of data. In the following section, an approach for dealing with a larger number of triples will be presented.

4.2 Clustering based on Fingerprints

For the second similarity measure we take an arbitrary hash function which maps the URI strings to 32-bit

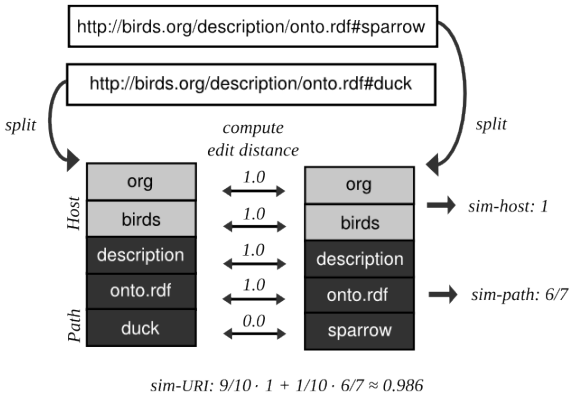

Figure 1: Example for the comparison of two URIs with the URI similarity measure.

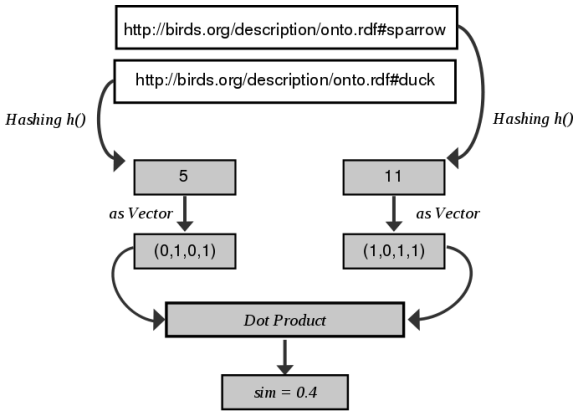

Figure 2: Example for the comparison of two URIs with the fingerprint similarity measure.

numbers and consider them as vectors as defined in equation 5.

$$\overrightarrow{h}(v) = (h_0(v), \cdots, h_i(v), \cdots, h_{31}(v)) \quad \text{with } h_i(v) = \begin{cases} 1, \text{if } h(v)/2^i \geq 1 \\ 0, \text{else} \end{cases} \tag{5}$$

As similarity measure sim_{vec} between two URI strings v and w we take the scalar product of their so called hash vectors which are normalized by the Euclidean Norm $\|\|$.

$$sim_{vec}(v,w) = \langle \frac{\overrightarrow{h}(v)}{\|\overrightarrow{h}(v)\|}, \frac{\overrightarrow{h}(w)}{\|\overrightarrow{h}(w)\|} \rangle = \frac{\sum_{i=0}^{31} h_i(v) \cdot h_i(w)}{\sqrt{\sum_{i=0}^{31} h_i(v)^2} \cdot \sqrt{\sum_{i=0}^{31} h_i(w)^2}} \tag{6}$$

The sim_{vec} value is in the range of 0 and 1. It is 1 if the hash values of the two URI strings are equal and 0 if the resulting hash vectors are orthogonal as the scalar product of two orthogonal vectors is always 0. For example, to compare two URIs u_1=*http://birds.org/description/onto.rdf#duck* and u_2=*http://birds.org/description/onto.rdf#sparrow*, these would first be mapped to 32-bit numbers by an arbitrary hash function h. For simplification, in this example we assume that the function maps to 4-bit numbers. Given that h(*http://birds.org/description/onto.rdf#duck*) = 5 = (0,1,0,1) and h(*http://birds.org/description/onto.rdf#sparrow*) = 11 = (1,0,1,1) the fingerprint-based similarity sim_{vec} between the two URIs would be ≈ 0.4. Figure 2 shows the calculation steps for this example.

For any scalar product holds:

$$\sum_{i=1}^{n} \langle x, y_i \rangle = \langle x, \sum_{i=1}^{n} y_i \rangle \tag{7}$$

Therefore for the vector similarity sim_{vec} the following holds.

$$\sum_{i=0}^{31} sim_{vec}(v, w_i) = sim_{vec}(v, \sum_{i=0}^{31} w_i) \tag{8}$$

This helps to reduce the computational effort of the `w`. For each r-cluster on a node with $r \in \{s, p, o\}$ we keep the sum of the hash vectors to all cluster resources $l_i \ldots l_n$ in that cluster up to date: $T_r = \sum_{i=1}^{n} h(l_i)$. An ant which wants to compute the similarity sum *Sim* for a r-cluster gets along with one comparison: $sim_{vec}(cr, T_r)$ instead of computing the similarity to all cluster resources $l_i \ldots l_n$ in that cluster: $\sum_{i=1}^{n} sim_{vec}(cr, l_i)$. As T_r is only updated when there are triples removed or added to the cluster, the effort to keep T up to date is relatively low compared to the computational costs that are omitted with each step that the `writing` ants make in the network. Therefore the computational time can be kept in $O(1)$ in opposite to the $O(n)$ time for the URI measure.

On the other hand, unlike the URI-based similarity measure, the vector-based similarity measure does not take any semantic similarity of the URIs into account. Depending on the given hash function, two URIs which are semantically similar can have completely different hash values or completely different URIs can have the same hash value.

5 IMPLEMENTATION

The RDFSwarms system is implemented in Java. It consists of an arbitrary large number of identical nodes following the P2P-paradigm.

Nodes and clients communicate over TCP/IP streams through an byte based protocol, where each operation is encoded as a single message. If a message is received by a node, a thread is started to per-

form the operation (e.g. reading from a store, updating the pheromone table etc.), migrating the thread from node to node until it fulfills its task. Ants are realized as single operations moving from node to node.

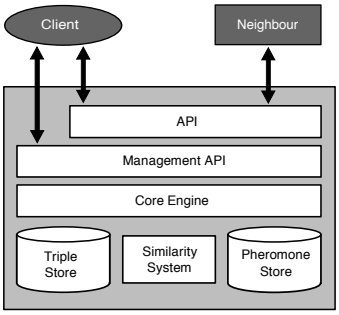

Figure 3: Architecture of the RDFSwarms System.

A node consists of several subsystems shown in figure 3. The external socket to which other nodes and clients can connect to is represented in the figure as outgoing arrows. Over these connections the ManagementAPI and the API communicates with the neighborhood and the clients.

Decoded messages are passed by the API to the Core Engine, which handles the management of the performing threads. Every outgoing operation is initialized by the Core Engine and is passed to the corresponding APIs which handle the communication.

Ant-based operations are forwarded to the modular similarity system, supporting the measures presented in section 4. It further provides an interface making the measures completely exchangeable for further research.

Furthermore the system contains a Sesame 2[5] based component for storing triples and pheromones. The storage system for triples is further divided into three sub-stores (see section 3). Each store is responsible for reading and writing triples according to its cluster type. For pheromones only one store exists, which is used by all ant-operations regardless of their type of cluster resource.

6 EVALUATION

For the evaluation we used four networks with respectively 10, 20, 30 or 40 virtual nodes which each had the topology of power law graphs. The test runs were executed on 10 identical machines, each equipped with 512 MByte RAM, a 2.4 GHz Intel P4 using Debian Lenny as operating system.

We made sure, that no connected nodes run on the same machine, otherwise the results would be falsified as the influence of the communication between the two nodes running on the same machine would not accounted into the evaluation results. As test data, several testing sets with respectively 1.000, 10.000, 100.000, 1.000.000 RDF triples randomly selected from the WordNet ontology[6], were used. After writing the data into the network, 10000 queries for the corresponding datasets were performed. The same test runs were carried out using the URI similarity as well as the fingerprint based similarity-measure. The averaged results per performed operation of the four test runs are shown in figures 4, 5 and 6.

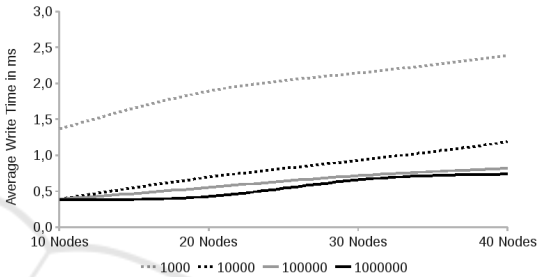

Figure 4: Average writing time using fingerprints.

Figures 4 and 5 show the time needed for writing n triples into a network of N nodes. The URI-similarity measure was only evaluated with datasets smaller than 10000 triples. As more nodes are added to the network, the stores are stronger partitioned, holding a smaller set of triples which can be compared faster with r when using the URI-similarity. The fingerprint similarity does not profit on more nodes, since *Sim* can always be computed the same time. The extra nodes visited in a larger network imply added costs, but the path length is still limited to the dimension d of the network, which is $O(log(N))$, because of the power law structure of the network.

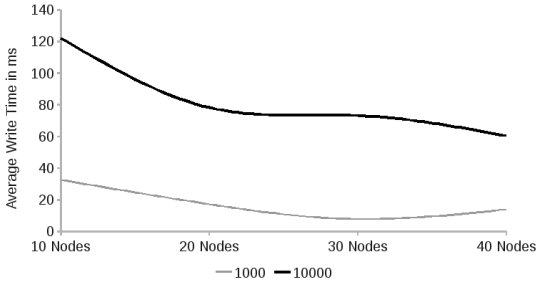

Figure 5: Average writing time using URI.

The scalability behavior over the amount of triples

[5]http://www.openrdf.org/

[6]http://www.w3.org/2006/03/wn/wn20/

differs between the two measures. Optimizations performed on the fingerprint measure realized a constant cost for comparing a cluster resource with its corresponding cluster. It even benefits from large amount of data, since the probability of finding a matching cluster rises with large amounts of data stored on each node. For the URI-clustering a similar optimization was not found so far. Performance tests show, that the fingerprint based system performed about 500 times faster then the URI system, storing about 2000 triples per second making it competitive to a local instance of a Sesame store in writing performance.

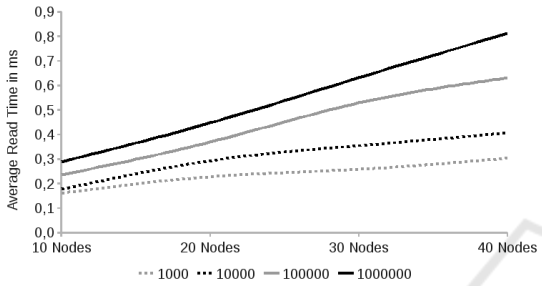

Figure 6: Average reading time per Triple.

Figure 6 shows the averaged test results for `read`. In opposite to `write`, `read` performs slower, as more triples are stored in the network, which can be justified with the scalability problems of the Sesame storage system (100 triples: 0.184 ms/triple, 10000 triples 3.434 ms/triple).

7 CONCLUSIONS AND FUTURE WORK

We have presented an approach to build scalable distributed RDF stores. At the core are basic clustering algorithms which are derived from ant colony optimization and configured by distinct similarity measures. In this paper we have presented two simple measures that can be determined by strictly local calculations.

We have further noticed the influence of the storage layer on the read performance. While being scalable on writing, the read performance drops, if enough data is stored. With a more scalable system, the network can process read queries for larger datasets in a shorter time. A new storage layer, which supports the measures natively would increase the whole performance, since current opererations suffer from a wrapping mechanism for Sesame.

As previously pointed out, our URI and fingerprint similarity measures bear a trade-off of performance against expressiveness, and vice versa, respectively. In the future we aim for a hybrid measure, that combines both clustering measures introduced here, enabling efficient RDF triple clustering with regard to URI similarities.

On another note, we are currently investigating concepts that allow for reasoning in such a RDF store by defining algorithms for reasoning ants. These then have to be integrated in our implementation and an optimization cycle of the engineered RDFSwarms implementation will be started. Furthermore, the RDFSwarms system will be applied as a storage infrastructure for an upcoming project dealing with semantic geoinformation.

REFERENCES

Bonabeau, E., Dorigo, M., and Theraulaz, G. (1999). *Swarm Intelligence: From Natural to Artificial Systems*. Santa Fe Institute Studies in the Sciences of Complexity Series. Oxford Press.

Cai, M. and Frank, M. (2004). Rdfpeers: A scalable distributed rdf repository based on a structured peer-to-peer network.

Gelernter, D. (1985). Generative communication in linda. *ACM Transactions on Programming Languages and Systems*, 7:80–112.

Koske, S. (2009). Swarm Approaches for Semantic Triple Clustering and Retrieval in Distributed RDF Spaces. Technical Report B-09-04B, FU Berlin, Institut für Informatik.

Menezes, R. and Tolksdorf, R. (2003). A new approach to scalable linda-systems based on swarms. In *Proceedings of ACM SAC 2003*, pages 375–379.

Nejdl, W., Wolf, B., Qu, C., Decker, S., Naeve, M. S. A., Nilsson, M., Palmer, M., and Risch, T. (2001). Edutella: A p2p networking infrastructure based on rdf.

Tolksdorf, R. and Augustin, A. (2009). Selforganisation in a storage for semantic information. *Journal of Software*, 4.