Multi-Level Indexing in a Distributed Self-Organized Storage System

Hannes Mühleisen, Tilman Walther and Robert Tolksdorf
Freie Universität Berlin – Networked Information Systems Group
Königin-Luise-Str. 24/26, D-14195 Berlin, Germany
muehleis@inf.fu-berlin.de, tilman.walther@fu-berlin.de, tolk@ag-nbi.de
http://digipolis.ag-nbi.de

***Abstract*—In many systems providing storage and retrieval operations on data, indices are used to make these operations more efficient. Distributed storage systems provide means to distribute the burden of storing and retrieving data onto multiple different computers. Routing indices can answer the central question in these systems: Where should one look for a specified data item? To be able to query for different columns in a relation or different entries in tuples, indexing for multiple dimensions is necessary. Our group applies a swarm-based approach to distributed storage leading to a new class of distributed systems, which are fully self-organized in their behavior and lack any shared global data structures. In this paper, we research whether multiple levels of routing indices can be maintained and used in such a distributed and self-organized storage service. To achieve this, we look into different types of indices and evaluate them in an experiment.**

I. Introduction

The efficient storage of large quantities of data is a crucial requirement for the operation of today's information systems. Many systems providing storage and retrieval operations for these quantities use indices in order to to support these operations efficiently. For example, a relational database can be configured to create an index on a specific column for a relation. Additional storage capacity and processing power is then spent to create and store a specialized access structure for this column for all entries in the relation. Once this is completed, queries including this column can be evaluated more efficiently during query processing. In general, indices are positioned in a trade-off between different types of efforts. The decision to spend resources in order to create an index is usually justified by vast resource savings during operation processing [1]. Indices depend on the data they have been created from: Should data items be removed or changed, indices containing these have to be updated as well.

Since the amounts of data to be stored by today's applications can easily outmatch the capabilities of single computers, distributed storage systems provide means to distribute the burden of storing and retrieving this data onto multiple different computers [2]. Two questions have to be answered by the entity controlling the storage and retrieval operations: Where should a data item be stored, and where should one look for a specified data item during retrieval. The answers to these questions are interleaved, and various different methods have been proposed to create these answers.

Peer-to-Peer (P2P) storage systems distribute not only the stored data, but also the control over operations onto multiple computers ("nodes"). Each node is capable of performing storage and retrieval operations on the entire set of data available on all nodes [3]. The means to achieve this differ: In most widely used approaches a synthetic network structure is constructed between the nodes to provide an access path, in others simple broadcasting methods are used. Routing indices can also be constructed in a distributed way and are used in various P2P systems to locate stored data.

To remove the dependency on fixed overlay network structures, another approach is to use self-organization techniques found in nature, especially in some species of ants. These swarm-based approaches applied to distributed storage lead to a new class of distributed systems, which are fully self-organized in their behavior and lack any shared global data structures [4]. Similar to P2P systems, operations can be initiated by any node, and the set of data items stored is distributed over the nodes. Trails of "virtual pheromones" are used to route queries to the node where relevant data is stored. These trails are created on-demand for every successful operation, and then speed up subsequent requests. Therefore, these pheromone trails can be compared to an index structure as described above: Their creation requires some effort, and they are used to speed up storage operations.

Many distributed storage systems such as key/value stores only support one single index on a single attribute for data stored. This enables them to use comparably simple and well-developed routing algorithms for their storage and retrieval operations. Other classes of systems, however, require more than one distributed index. Examples for these classes include distributed relational databases or distributed tuple spaces. To be able to query for different columns in a relation or different entries in tuples, indexing for multiple dimensions has to be supported. Additionally, the comparison operator used on the index may not always be the "equals" operator, range and distance queries using arbitrary metrics are also required by many applications.

For this paper, our central research question is therefore: How can multiple levels of routing indices be maintained and used in a distributed, swarm-based, and self-organized storage service?

The remainder of this paper is structured as follows: We begin with providing an overview over related work both on distributed indices as well as on self-organized systems in Section II. The concepts and algorithms behind our Self-Organized Semantic Storage Service are outlined in Section III, followed by different types of indices to be considered. We then describe our evaluation methods and results in Section V and finally conclude this paper in Section VI.

II. Related and Previous Work

While indices have been used by relational databases for decades [1], the index architectures described in this paper are not focusing on speeding up the access to data stored in the secondary storage of a single computer, but rather aiding operation routing decisions in a distributed storage system. Early systems such as *Napster* used central "index" servers to determine the nodes storing data relevant to a query [5]. Later approaches did not rely on these central systems to remove a single point of failure. There, the processing of storage and retrieval operations relies on proper query routing methods, and routing indices can be used to improve routing accuracy and efficiency. Distributed query processing with its selection of multiple participating sites, cost estimation, and query optimization can then be performed on top of these routing methods [6].

A. Routing Indices

Crespo and Garcia-Molina described the concept of routing indices, which are used to forward single-key queries to neighboring nodes within the network which are more likely to produce a result [7]. This can be compared to the Distance-Vector method used to calculate routing tables in computer networks. Indices are created as connections to other nodes are established, which leads to high reorganization efforts for unstable networks. This approach has been extended by Karnstedt et al. for more complex XML data structures and corresponding queries [8]. Brunkhorst et al. further expand the concept with the distinction between different classes of nodes and different routing index granularities [9]. They especially assume a well-defined structure within the stored data, such as RDF. Examples for a distributed P2P system to execute retrieval operations on RDF data are the *Edutella* system by Neidl et al. [10] or *YARS2* by Harth et al. [11].

B. Self-Organized Storage Systems

Swarm intelligence has been identified to be a powerful family of methods by Bonabeau et al. [12]. Menezes and Tolksdorf applied swarm intelligence to a distributed tuple space built to implement the Linda coordination model [13], [14]. This space can also be regarded as a self-organized distributed storage system. They introduced basic concepts of ant colony algorithms that are suitable for tuple storage and retrieval [13]. Following a trend of trading away soundness and completeness to gain efficiency in the Semantic Web area as summarized by Fensel et al. [15], Tolksdorf and Augustin then applied this idea to distributed RDF storage and used a syntax-based similarity metric to cluster syntactically similar resources on neighboring storage nodes. Their concept was successfully evaluated using simulation runs [16]. A similarity metric based on semantic similarity measures and aimed at achieving the clustering of related concepts was introduced in [17]. Furthermore, we have contributed enhancements to the concept towards an full implementation of a distributed *Self-Organized Semantic Storage System* (S4) and also presented preliminary evaluation results using the target implementation [4]. We will introduce our approach, its routing methods and index structures in the following section.

III. Ant Behaviour for Distributed Storage

As our group has shown in previous research, the behaviour found in several ant species to forage for food has been found suitable to implement a scalable distributed and self-organized storage system [4], [18]. Here, ants on the search for food start off from their nest on a random walk on the landscape. They continue on this random walk until they encounter food. A portion of the food found is then picked up and carried back to the nest, which is located using its distinct nest scent. On the way back to their nest ants leave a trail of chemical pheromones behind, which form a pheromone path from the food source to the nest. Now, if other ants on their random walk in search for food encounter this pheromone trail, they are inclined – albeit not forced – to follow it in the direction of the food. Should they also find food following this path, they will further increase the pheromone intensity on their way back, also increasing the inclination of other ants to also follow this path. As pheromone trails also evaporate over time, old trails for example leading to depleted food sources are falling out of favor quickly. From an Artificial Intelligence perspective, this represents a positive enforcement mechanism between large numbers of independent simple agents [19].

A. Storage System Concept

Our Self-organized Semantic Storage Service (S4) provides a storage system based on the described foraging algorithm. The data in this store is distributed over a network of homogeneous nodes, each of which provides access to the entire store. There is no central instance monitoring the entire network, all decisions are based on the information that is locally available. This decentralized organization enables the system to offer a high degree of scalability and robustness. In our system, a number of computers ("nodes") interlinked by network connections are regarded as a landscape for virtual ants. Each node is connected to a limited set of other nodes, and stores a portion of the data to stored in the entire system. Each node offers an interface for client applications, able to store and retrieve data items. Storage and retrieval requests are modeled to be virtual ants able to autonomously move about on that landscape, typically requiring multiple hops over network connections between nodes in order to find a data item. On each node, requests compute the node to be visited next from the nodes directly connected to the current node using virtual pheromone trails. These virtual pheromone trails are laid tracing

back the path successful operations, aiding current and future operations in their routing decisions. A small random factor is also included in those routing decisions, modeling the ant's inclination to follow pheromone paths. As a replacement for a nest scent, operations carry a path history of visited nodes. If requests are successful, this path is traced back and the virtual pheromones are intensified, yielding the described positive enforcement for each stored data item.

The entire system is able to globally optimize request routing for all stored data items, given a sufficient number of subsequent requests for a data item [20], [4]. One main benefit is the complete in-transparency of the actual location of stored data inside this system. This allows the system to adapt very easily to changing data characteristics and fluctuations in data popularity.

Data is written to the store in form of tuples, ordered lists of values with a predefined number of elements. The most common type of tuples used in context of the system are RDF triples as defined by the Resource Description Framework proposed by the W3C for the expression of semantic metadata. However, the system is not limited to the storage of semantic information since it can process any kind data that is provided as tuples.

B. Data and Index Updates

The second central concept for the storage and retrieval of tuples is the local concentration of similar tuples as clusters on the same or a neighboring node. This is required to avoid ambiguity in the pheromone paths. When written to the store, a single tuple is processed by a similarity function that maps the tuple's value to a discrete range of numerical values, the *cluster key*. This value is then being used for the clustering of the stored data: Write operation moves tuples through the storage network, searching for a cluster that fits their value. On every node, the cluster key is being compared to the reference values of the clusters stored on the node. Once the fitting cluster is reached, the tuple is stored on the node and a virtual pheromone trail is laid out on the edges of the network, back to the node the tuple originated from. Should no fitting cluster be found within a given limit, the tuple is simply stored on the node the write operation is currently visiting.

Using this technique, a data structure is generated that leads the way to each particular cluster in the network. As the virtual pheromone trails are updated constantly with every read and write operation, this structure always represents the current state of the clusters. In other words, the virtual pheromones constitute a routing index on the data in the store which can be compared to the approach presented by Crespo and Garcia-Molina [7].

Since the virtual pheromone trails evaporate over time, the index is constantly updated by the read and write operations. The new pheromones that represent the current structure of the stored data replace the outdated information. This evaporation is performed on every read or write access on the index.

To retrieve stored tuples, a retrieval template is defined. This template consists of a required entry for tuples to match. For example, a tuple (a, b, c) could be retrieved by using a template of the form (a, *, *) with * being a wild card for any possible value. Using this template, a read operation is requested on an arbitrary node in the storage network. The node creates the retrieval operation which first checks the local storage and – should there be no match – is being forwarded to the neighbor node most likely to contain or lead to the matching cluster. This likeliness is calculated using virtual pheromones stored on the connections to the neighboring nodes. These pheromones express the contents of the clusters which can be found in the part of the network reachable by the corresponding connection.

As the amount of cluster values is directly dependent on the number of elements stored in the system and the pheromone table has to be maintained on every node, this would present a scalability problem. Therefore the virtual pheromones on the connections to the neighboring nodes are aggregated as a range of cluster key values. Every range is defined by four values; minimum, maximum, average and element count. The number of ranges stored on a particular connection is limited, thus keeping the space used by this data structure at a constant level.

IV. Routing Indices

In this section, we will investigate the explicit requirements to create and maintain such an index as well as the possibilities for additional levels of indexing. As we have shown in the previous section, the virtual pheromone trails along with the stored elements constitute a routing index for a single entry of the stored tuples. Queries using keys other than the configured and present one are not supported in this setting due to missing pheromone trails for any other key. If the first tuple entry is configured to be relevant for tuple storage, queries using – for example – the second tuple entry would require a "full network scan", where every node is queried for matching data items

To support retrieval and storage operations for a particular entry on the stored tuples, the following requirements have to be met:

- *Key definition* – The tuple entry relevant for the generation of cluster keys used in the routing process
- *Similarity Measure* – A method to convert a tuple entry into a cluster key on a continuous and limited scale. In our context, similarity measures map tuple entries to floating-point values in the interval [0,1]. Similarity calculation has to be performed independently on every node, thus all required information should be present on the current node.
- *Pheromone Path Layer* – For every index, pheromone information has to be kept on every node. However, due to the limitation of pheromone data stored as described in the previous section, the additional overhead for new pheromone path layers is limited and grows only linearly.
- *Independent Storage* – As only the cluster key determines the location of a tuple in the network, the tuple has to be stored on the node the pheromone path leads to. This requires independent storage for additional index levels, as different indexing levels can lead to different storage locations for a single tuple. Hence, every new index

requires its own independent storage layer and cluster set throughout the entire network. This constitutes the main trade-off for additional indices, and costs and benefits have to be weighted against each other.

The costs of maintaining a single index thus consist of the storage required to keep the pheromone paths as well as the storage required to provide the independent storage layer. As a simplification, the storage space needed to store a single tuple or primitive value is assumed to be 1. For e tuples to be stored in a network of s nodes, a neighbor connection count of n and a range limit on the neighbor connections of c, and a range definition size of 4 this evaluates to

$$e_s := e + s \times n \times c \times 4$$

This can now be considered the general cost to construct an additional index level within the S4 system, provided a compatible similarity measure can be constructed. We have identified a special case for additional indices: If an additional index is using the same similarity measure as another index, they can share a pheromone path layer. For example, if a tuple with three string entries (as in RDF) is to be indexed using all entries as indices, the additional two index layers require $2 \times e$ storage space.

A. Heterogeneous Indices

In this section, we describe a number of possible additional index types along with concrete similarity measures where applicable. This is performed to determine whether the indexing of a particular type of data is feasible in the context of our S4 concept. If a concrete similarity measure can be given, we consider the specific dimension to be fitting for the generation of indices within our concept.

1) String-based: Text strings can be mapped to a numeric value using an arbitrary hash function. For example, the SHA1 hashing function [21] produces hashes with the length of 160 bits. To generate an cluster key c_k from a string s using SHA1, the following formula may be used:

$$c_k := \text{SHA1}(s)/2^{160}$$

If range queries are to be supported, locality-preserving hash functions [22] can be employed. In this case, a length limit for the relevant parts of the string to be hashed has to be instated.

2) Temporal: To create an index supporting the retrieval of tuples according to temporal constraints, the similarity measure has to map the time to be indexed to a different interval. In order to perform this, minimum and maximum values for the source interval have to be defined. If we assume temporal information to be given as timestamps t, and we define t_{min} to be the minimum and t_{max} as the maximum value for indexed temporal data, the following similarity measure may be used:

$$c_k := \frac{t - t_{min}}{t_{max}}$$

3) Spatial: If we assume spatial information is not defined as a hierarchy of places, but rather as a numerical representation in a two-dimensional coordinate system (e.g. partial WGS84-Coordinates [23]), the definition of a similarity measure for spatial indexing is also straightforward. The product of the coordinate values x and y can simply be normalized to the interval [0,1].

4) Semantic: If the storage system is used to process semantic information (e.g. RDF triples annotated using OWL ontologies), similarity calculation is more complex. The presence of semantic annotations would enable sub-class matching for queries. For example, the tuples (brian, is-a, dog) and (dog, is-a, mammal) can be used to derive the fact that the entity "brian" is an instance of the concept "mammal". An application could now present the query for all "mammals", which – due to the information inherent in the stored tuples – would also apply to the entity "brian". In order to create and query an index for this kind of information, the similarity measure needs both the instance and the ontology information to be present. Tolksdorf et al. have presented such an approach, however, at least parts of the ontology information have to be carried with the retrieval operations [17]. Therefore, this indexing is more costly than the previous ones and benefits have to be weighted against the costs more carefully. A different approach to this issue is to materialize all implicit information during periodic operations, which has been proposed by Obermeier et al. [24].

V. Experimental Results

We have performed a number of experiments to determine the impact the generation of routing indices has on the performance and behavior of our S4 system. We have measured the write time as well as the amount of stored elements for each node for different network sizes and different index configurations. Network sizes ranged from 10 to 150 nodes, and one, two and three string indices were configured on each node for the different test runs. From our theoretical observations we expect increasing write time and element count for additional indices. For larger network sizes with a constant amount of stored triples, write time is expected to increase modestly, while element counts per node are reduced.

We have implemented the aforementioned concepts of the S4 system and its practical enhancements such as clustered similarity values using the Java programming language as a stand-alone system. This implementation was deployed onto our test cluster consisting of 150 virtual Linux nodes running on a single server with eight 2.26 GHz processors and 64 GB of main memory. For a typical run, a node had on average 11 neighboring nodes. A random subset of the DBpedia dataset [25] containing 100.000 RDF triples was stored in the storage network for each test run.

A. Write Time Impact

For each network and index configuration, our DBpedia subset consisting of 100K triples was written to a single arbitrary node, on which the operations distributing and clustering the data inside the network were dispatched. The

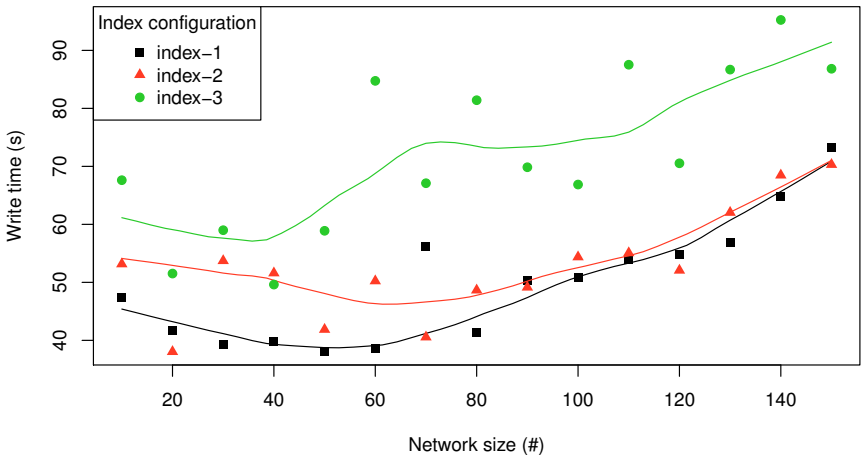

Fig. 1. Write time for storing 100K triples per network size and index configuration

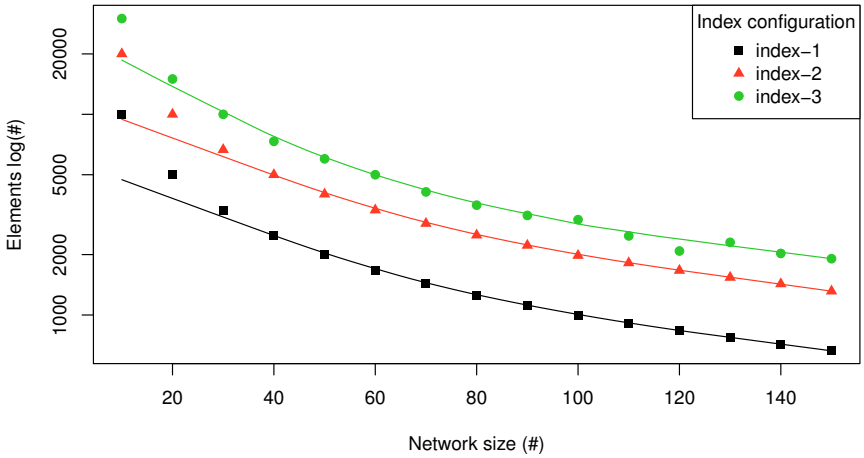

Fig. 2. Average element count after storing 100K triples per network size and index configuration

time required to complete the write operation from the client's standpoint was recorded for each run. Figure 1 shows a scatter plot and LOESS nonparametric regression lines [26] of these recordings for the different network sizes (horizontal axis) and the time required (vertical axis). Point styles describe the results for different index configurations. Significant fluctuations of the results can be observed, which we consider an effect of the system-inherent randomness as described in Section III. However, it is also visible that the tree-fold index (● markers) generally longer to generate than the two-fold index (▲ markers), which in turn took longer than the one-fold index (■ markers). Also, no exponential behavior is observable, we therefore assume sufficient scalability of our indexing method for additional indices and nodes.

B. Element Overhead

Since the generation of additional routing indices (or pheromone trails) leads to additional copies of the stored elements to be written to single nodes, we have measured the average number of elements stored on the nodes for different network sizes and index configurations. These measurements are given in Figure 2. As expected in the previous chapter, the average number of elements stored on each node for different network sizes increases linearly.

VI. Conclusions and Future Work

After introducing how indices are used to help operation execution for any system storing bulk data, we have described the need for distributed storage to overcome scalability limitations. By differentiating between central and distributed indices in their system settings, the need for distributed indices became apparent. We compared the distributed indices proposed by other researchers, especially "routing indices" with their various levels of granularity.

The main focus of this paper was the question whether multiple levels of routing indices can be maintained and used in a distributed, swarm-based, and self-organized storage service. Each index can be used to efficiently answer a distinct set of queries within the system, and support for multiple indices thus enhances the overall usefulness of our system.

Our swarm-based self-organized approach to distributed storage ("S4") was introduced along with its comparable

concept of indexing and routing paradigm. The conceptual requirements to create a usable routing index within our system were described, and from there various different types of indices to support a wide range of queries were examined. Our theoretical examination showed the general feasibility and theoretical effort to build multi-leveled indexing in this specific class of systems, and also showed the advantages of indices not relying on additional information.

We have then performed an evaluation using experiments on our test cluster consisting of 150 virtual machines and our S4 implementation. Evaluation showed the expected behaviour from our self-organized system, where additional indices represented a linear overhead. This is consistent with our expectations, and we thus are inclined to answer our research question positively. This paper has contributed an theoretical insight into the feasible type of indices in this class of distributed systems as well as an evaluation of their overhead.

A. Future Work

For our future work, we would like to implement the mentioned index variants in a scalable, self-organized way. A flexible and distributed configuration mechanism for the indices is also necessary, as all nodes in a storage network should have a compatible understanding of the configured index structures. We have shown how indices operating on similar data can share a routing index structure, but it could be feasible to also integrate heterogeneous data into the same index structure. Also, the movement of elements in the system with the goal of reducing the routing indices' size could also be a promising approach to achieve index compression.

Query processing is also a very interesting topic in this area, we shall investigate how queries which have to make use of more than one of these indices could be evaluated efficiently. Classical query optimization for distributed systems use a cost model and cardinality statistics for query optimization [6]. It may be possible to achieve a similar efficiency in a self-organized system. For evaluation purposes we are planning to extend our testbed network to larger storage networks, possibly exceeding 1000 "real" participating nodes for our experiments. Experiments using more than three configured indices are also a target for further work as well as retrieval experiments, which show the advantage gained by additional indices weighted against the effort of building them.

ACKNOWLEDGMENT

This work has been partially supported by the "DigiPolis" project funded by the German Federal Ministry of Education and Research (BMBF) under the grant number 03WKP07B.

REFERENCES

[1] Bayer, R. and McCreight, "Organization and maintenance of large ordered indexes," *Acta Informatica*, vol. 1, pp. 173–189, 1972.

[2] P. N. Yianilos and S. Sobti, "The evolving field of distributed storage," *IEEE Internet Computing*, vol. 5, no. 5, pp. 35–39, 2001.

[3] R. Schollmeier, "A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications," in *Peer-to-Peer Computing*, R. L. Graham and N. Shahmehri, Eds. IEEE Computer Society, 2001, pp. 101–102.

[4] H. Mühleisen, A. Augustin, T. Walther, M. Harasic, K. Teymourian, and R. Tolksdorf, "A self-organized semantic storage service," in *Proceedings of the 12th International Conference on Information Integration and Web-based Applications & Services (iiWAS2010)*, 2010.

[5] B. Yang and H. Garcia-Molina, "Comparing hybrid peer-to-peer systems," in *Proceedings of the Twenty-seventh International Conference on Very Large Databases*, 2001.

[6] D. Kossmann, "The state of the art in distributed query processing," *ACM Computing Surveys*, vol. 32, no. 4, pp. 422–469, 2000.

[7] A. Crespo and H. Garcia-Molina, "Routing indices for peer-to-peer systems," in *Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02)*, ser. ICDCS '02. Washington, DC, USA: IEEE Computer Society, 2002, pp. 23–.

[8] M. Karnstedt, K. Hose, and K.-U. Sattler, "Distributed query processing in p2p systems with incomplete schema information," in *Proceedings of CAiSE'04 Workshops, 3. Int. Workshop on Data Integration over the Web (DIWeb2004 icw CAiSE)*, 2004, pp. 34–45.

[9] I. Brunkhorst, H. Dhraief, A. Kemper, W. Nejdl, and C. Wiesner, "Distributed queries and query optimization in schema-based P2P-systems," in *In Proceedings of the International Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P*, 2003, pp. 184–199.

[10] W. Nejdl, B. Wolf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, M. Palmr, and T. Risch, "EDUTELLA: A P2P networking infrastructure based on RDF," *Proceedings of the eleventh international World Wide Web Conference*, Jan. 01 2002.

[11] A. Harth, J. Umbrich, A. Hogan, and S. Decker, "YARS2: A federated repository for querying graph structured data from the web," in *The Semantic Web, ISWC 2007, Busan, Korea, November 11-15, 2007*, ser. Lecture Notes in Computer Science, vol. 4825. Springer, 2007, pp. 211–224.

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

[13] R. Menezes and R. Tolksdorf, "A new approach to scalable linda-systems based on swarms," in *Proceedings of ACM SAC 2003*, 2003, pp. 375–379.

[14] D. Gelernter, "Generative communication in Linda," *ACM Transactions on Programming Languages and Systems*, vol. 7, pp. 80–112, 1985.

[15] D. Fensel, F. van Harmelen *et al.*, "Towards larKC: A platform for web-scale reasoning," in *ICSC*. IEEE Computer Society, 2008, pp. 524–529.

[16] R. Tolksdorf and A. Augustin, "Selforganisation in a storage for semantic information," *Journal of Software*, vol. 4, 2009.

[17] R. Tolksdorf, A. Augustin, and S. Koske, "Selforganization in distributed semantic repositories," *Future Internet Symposium 2009 (FIS2009)*, 2009.

[18] H. Mühleisen, T. Walther, A. Augustin, M. Harasic, and R. Tolksdorf, "Configuring a self-organized semantic storage service," in *Proceedings of the 6th International Workshop on Scalable Semantic Web Knowledge Base Systems*, 2010, pp. 1–16.

[19] M. Dorigo and T. Stützle, *Ant Colony Optimization*. Cambridge, Massachusetts: The MIT Press, 2004.

[20] R. Menezes and R. Tolksdorf, "A new approach to scalable linda-systems based on swarms," in *Proceedings of ACM SAC 2003*, 2003, pp. 375–379.

[21] D. E. Eastlake and P. E. Jones, "US secure hash algorithm 1 (SHA1)," Internet RFC 3174, Sep. 2001.

[22] A. Chin, "Locality-preserving hash functions for general purpose parallel computation," *Algorithmica*, vol. 12, no. 2/3, pp. 170–181, Aug./Sep. 1994.

[23] National Imagery and Mapping Agency, "Department of defense world geodetic system 1984," NIMA TR8350.2, Third Edition, Jan. 2000.

[24] P. Obermeier, A. Augustin, and R. Tolksdorf, "Towards swarm-based federated web knowledgebases," in *Proceedings of the Workshop on Self-Organising, Adaptive, Context-Sensitive Distributed Systems (SAKS-2011)*, 2011.

[25] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives, "DBpedia: A nucleus for a web of open data," in *6th International Semantic Web Conference (ISWC 2007)*, ser. Lecture Notes in Computer Science, K. Aberer, K.-S. Choi, N. F. Noy *et al.*, Eds., vol. 4825, Busan, Korea, Nov. 2007, pp. 722–735.

[26] W. S. Cleveland, "Robust locally weighted regression and smoothing scatter plots," *Journal of the American Statistical Association*, vol. 74, no. 368, pp. 829–836, 1979.