The current issue and full text archive of this journal is available at
www.emeraldinsight.com/1744-0084.htm

A survey on self-organized semantic storage

Hannes Mühleisen, Tilman Walther and Robert Tolksdorf
Network-Based Information Systems Group, Department of Computer Science, Freie Universität Berlin, Berlin, Germany

Received 23 May 2011
Revised 25 May 2011
Accepted 26 May 2011

Abstract

Purpose – The purpose of this paper is to show the potential of self-organized semantic storage services. The semantic web has provided a vision of how to build the applications of the future. A software component dedicated to the storage and retrieval of semantic information is an important but generic part of these applications. Apart from mere functionality, these storage components also have to provide good performance regarding the non-functional requirements scalability, adaptability and robustness. Distributing the task of storing and querying semantic information onto multiple computers is a way of achieving this performance. However, the distribution of a task onto a set of computers connected using a communication network is not trivial. One solution is self-organized technologies, where no central entity coordinates the system's operation.

Design/methodology/approach – Based on the available literature on large-scale semantic storage systems, the paper analyzes the underlying distribution algorithm, with special focus on the properties of semantic information and corresponding queries. The paper compares the approaches and identify their shortcomings.

Findings – All analyzed approaches and their underlying technologies were unable to distribute large amounts of semantic information and queries in a generic way while still being able to react on changing network infrastructure. Nonetheless, as each concept represented a unique trade-off between these goals, the paper points out how self-organization is crucial to perform well at least in a subset of them.

Originality/value – The contribution of this paper is a literature review aimed at showing the potential of self-organized semantic storage services. A case is made for self-organization in a distributed storage system as the key to excellence in the relevant non-functional requirements: scalability, adaptability and robustness.

Keywords Semantic web, Distributed storage, Self-organization, Peer-to-peer, Computer software, Semantics

Paper type Literature review

1. Introduction

The semantic web has provided both researchers and developers with a vision how to build the applications of the future. Within these applications, semantics facilitate easy data integration, and support the generation of new knowledge by inferring from already present data. Around applications a new ecosystem of supporting systems and technologies unfolds, and while many challenges have already being tackled, there is still room for improvement in many areas. If a semantic web application is not only consuming

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. The authors would like to thank student assistants Alexander Dümont, Dennie Kabul and Sascha Gennrich for their efforts in researching and proofreading this paper.

International Journal of Web Information Systems
Vol. 7 No. 3, 2011
pp. 205-222

1744-0084
DOI 10.1108/17440081111165866

data, but also publishing semantic information, the need for a software component dedicated to the storage and retrieval of semantic information becomes immanent. In the past, many applications used a dedicated relational database to store and retrieve data. While it is still possible to use a relational database to store and retrieve semantic web data, custom-built solutions can offer both better performance as well as support for semantic web data formats and query languages.

An effort headed by the World Wide Web Consortium (W3C) has lead to a number of standards regarding semantic web data. The support of these standards has become the *de facto* functional requirement for semantic storage systems. The resource description format (RDF) has been standardized as a directed graph with well-defined semantics, which can be exchanged in one of its multiple serializations, for example RDF/XML or RDF/N3 (Klyne *et al.*, 2004; Hayes and McBride, 2004). Storage systems are required to store these graphs. In order to access the stored information, the SPARQL Query Language has been designed and standardized (Prud'hommeaux and Seaborne, 2008). Support for this language is thus also required from storage systems.

Apart from mere functionality, storage systems are also rated according to important non-functional requirements. They include scalability, which describes the behavior of the system when faced with increasing amounts of stored data or large numbers of queries in a given time. Also, since storage systems are usually off-the-shelf components, they have to exhibit adaptability towards the characteristics of stored data and queries not known beforehand. Finally, as the stored data are usually mission-critical for the institutions storing them, robustness towards failure of system components is also a central non-functional requirement.

The amount of data which can be expected from applications is getting larger and larger. Recently, Google's index contains over one trillion (10^{12}) publicly available web pages (Alpert and Hajaj, 2008), not including content that is hidden behind search forms. A side-effect from creating a global semantic web is to expose this so-called "deep web" to third-party systems, which is estimated to reach 500 times the size of the "visible" web (Lawrence and Giles, 1999). Handling these amounts of data is not feasible on stand-alone computer systems for economical reasons; a number of smaller machines can provide the computing and storage capacity of a larger machine for a fraction of the cost of one large machine following the shared-nothing paradigm. Hence, if the task of storing and querying these enormous amounts of data could be distributed onto many shoulders, querying the entire web could again become possible. For the related task of content distribution, the BitTorrent concept as an example has made content distribution to an arbitrary number of clients possible (Eubanks, 2005).

The distribution of a task onto a set of computers connected using a communication network is not trivial at all. The first approach would be to task a "special" machine to coordinate the actions of the entire set of computers (commonly called nodes), for example deciding on which computer a data item is to be placed, or how to evaluate a query for stored data items. These centralized or clustered approaches have been shown to scale and adapt well, and we introduce several examples for federated semantic storage services in Section 3. However, by relying on special nodes these systems exhibit a potentially fatal vulnerability. Should the node coordinating the storage operations fail, the entire storage system can no longer function, impeding its robustness.

The obvious solution to the vulnerability inherent in federated system is the elimination of special nodes from the set of nodes tasked to provide a service. In self-organized systems, every node takes a share in the responsibilities of the entire system, and sufficient redundancies reduce the likelihood of system failure and data loss to statistically acceptable figures. In the case of storage systems, there are two tasks to be distributed: first, the nodes have to be able to determine the node where a new data item should be placed, with each node being able to make this decision or contribute to a decision-involving multiple nodes. Second, nodes have to be able to locate data items stored in the past within the network. While both tasks could be achieved by simply flooding appropriate messages to every node participating in this so-called storage network, a scalable solution requires a more thoughtful approach. We will introduce the basic network organization patterns as well as systems providing a semantic storage service on top of such a network in Section 4.

The contribution of this paper is a literature review aimed at showing the potential of self-organized semantic storage services. We try to make a case for self-organization in a distributed storage system as the key to excellence in the mentioned non-functional requirements scalability, adaptability and robustness.

2. Distributed storage of semantic information

If data are to be stored within a network of multiple nodes, the shape and properties of the data to be stored as well as the dynamics of a distributed storage system have to be taken into account. In this section, we give an overview over general issues for distributed storage as well as an introduction into the standardized storage and query format for semantic information. The implications of the usage of this data format and query methods for distributed storage are also discussed.

2.1 General issues in distributed storage

Within a share-nothing distributed system, each participating node is dependent on the correct interaction of its various complex components. In many computer systems, most components represent single points of failure. This is particularly the case for systems used to build a cost-effective large-scale distributed system. Hence, the failure of a single component will cause the immediate unavailability of the entire node. While the failure probability of single nodes may be low, the probability of any node failing in a distributed system approaches one as the number of nodes increases. The distribution technology has to be able to handle these cases without seriously impeding the system's operation. Since nodes are connected by network technology, the failure of network infrastructure has also to be taken into account. Also, these failures easily affect a whole number of nodes, a failing switch for example can affect several nodes at the same time. An additional problem is created if the network connections are repaired, as the nodes then should resume their task as the part of a storage network. Equally, a robust distribution technology has to be able to handle all these cases.

The properties of the data items in the storage network are a reason for further challenges. For example, if the size of the stored data objects is not distributed evenly, a storage system may have to provide different strategies of handling these different sizes. Also, uneven data popularity can also be the root of issues: should few data items receive a majority of requests, and should these data items unfortunately be stored on the same physical machine, this machine becomes a bottleneck, and an adaptive

distributed system should able to balance out this load problem by, for example, moving some popular data items to idle nodes.

2.2 Data format and query language

The RDF (Klyne *et al.*, 2004) is an universal format to describe arbitrary resources with arbitrarily complex data structures. This is achieved by describing a directed graph with labels on its edges and nodes. Nodes are labeled using standard URIs, and nodes can either be also an URI or a literal value, e.g. an integer. These graphs can be decomposed into so-called triples, each consisting (in this order) of subject, predicate and object, denoting an arc in the graph from the node identified with the subject value to the node identified with the object value using the predicate value as property.

While the issue of indexing such a graph to support efficient query execution has also been discussed for graphs in general (Sakr and Al-Naymat, 2010), RDF's particularities provoked a more specific approach. The information stored in an RDF graph can be queried to answer a specific question, for example "How many people live in Berlin?". If we consider the simple RDF graph denoted by the triple (p:Berlin, p:hasInhabitants, 3400000), answering the query now requires the location of the node for p:Berlin, finding the arc with the label p:hasInhabitants, traversing it, and then reading the literal value of the node the arc leads to. This is concisely expressed in so-called basic graph patterns (BGP), which give a pattern to be matched against the entire graph. Graph patterns are a combination of wild cards with fixed values, e.g. (p:Berlin, p:hasInhabitants, *) with * as wild card. If variables are used for the wild cards, multiple triple patterns can be joined together for complex queries. This method of querying a graph has been standardized in the query language SPARQL (Prud'hommeaux and Seaborne, 2008).

RDF graphs can be enriched with defined semantics, where predicate values are assigned formal semantics (Hayes and McBride, 2004). While a number of predicate values are pre-defined and allow, for example, the definition of a type for a resource, users can always define new types and predicates using the RDF Schema language (Brickley and Guha, 2004). These so-called vocabularies are also encoded as RDF, and can be handled the same way as the graphs making use of them. However, the presence of these schemata can also allow the storage system to make more accurate assumptions about the properties of the data to be stored, and some storage systems thus require schema information to be pre-configured.

2.3 Format-specific issues

In addition to the general issues for distributed storage as pointed out in Section 2.1, the storage of RDF graphs creates a number of more specific challenges. These challenges make it difficult to design a storage scheme that can be shown to be efficient for all possible cases (Battré *et al.*, 2006). The usage of pre-defined vocabularies along with practicability reasons typically creates a very small frequency of values for the predicate entries, if the RDF graph is deconstructed into a list of triples. The even smaller set of standardized properties for example for resource type definition further exacerbate this problem. On the other hand, both subject and object entries describing the actual resources and literal values typically show an immense frequency of distinct URIs. Obviously, any storage or distribution strategy not distinguishing between the different frequencies would be sub-optimal.

From the requirement to support a query language based on BGP such as SPARQL as described in Section 2.2, the handling of graphs as atomic entities is impossible. Rather, efficient support for BGP requires indexing on the graph node or triple level. However, the storage space occupied by these indices can easily exceed the space required to store the serialized graph. For this reason, some RDF storage systems no longer store the verbatim triples, but only derived values inside their various indices. However, this approach is no silver bullet, since RDF graph nodes can also contain literals of a size unbounded by the standard documents (Klyne *et al.*, 2004).

Skewed data popularity within RDF graphs is also an issue, again due to the standardized and heavily used properties, for example for resource type definition. Experiments have shown that over 90 percent of queries use one of these properties, inevitably creating hot points within the data set (Harris *et al.*, 2009).

In the light of these challenges, it becomes clear that any storage system aimed at handling RDF graphs and corresponding queries needs to be designed carefully in order to answer all of these challenges. Otherwise, the system's capability to scale with regards to data and query load is jeopardized. In the following, we will introduce different approaches, and analyze how well they – in theory – handle these issues. The outcome of this analysis can give recommendations for the selection of a distributed storage solution for RDF as well as hints for further research and development.

3. Clustered storage

The first approach to the distributed storage of RDF graphs is to partition the graph onto many storage nodes. A central coordination node serves as a front-end to users executing their queries. Flooding each request to all storage nodes would not scale neither with the network size nor with the amount of queries per time unit. Hence, accessory data structures – for example the mapping of resource URIs to storage nodes – are kept on the coordinating node. Generally, all nodes in such a setup are under the control of a single organization, and the physical distance between the nodes is small. While nodes do not maintain shared data structures, only the task of storing data is distributed, with the control over the operation of the entire system staying centralized on the control nodes. Consistent with the literature, we refer to this structure as a centralized or clustered distributed storage system. In the following, we present and analyze a number of clustered storage systems that have been proposed, in an attempt to determine their degree of self-organization. Figure 1 shows this structure. A control node C1 handles requests from applications, it is directly connected to several storage nodes S1 through S6.

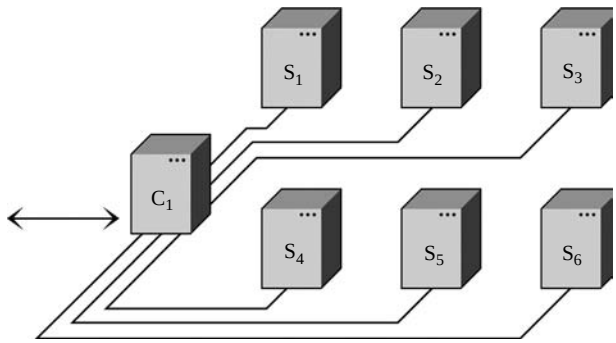

Figure 1.
Network structure for clustered storage

3.1 The clustered tuple database

The clustered tuple database (TDB) (Owens *et al.*, 2008) is based on the highly successful stand-alone on-disk RDF store TDB. Like many RDF stores, TDB first maps RDF nodes to so-called NodeIDs using their literal representation. Using these NodeIDs as key, three index structures are created for each triple component, each containing a copy of all triples. To be able to perform queries on the indices, a fourth index maps each node's literal representation towards a NodeID. Large literal values are stored inside an additional table, however, the system always tries to store fitting literals directly inside the index ("inlining").

Extending TDB in order to create a clustered storage system hence required to distribute all five data structures. In general, data structures are distributed using horizontal hash partitioning (DeWitt and Gray, 1992). Two groups of storage nodes are defined: a small group of control nodes ("query coordinators") handles queries, thus they each store a part of literal-NodeID mapping table as determined by hashing. Storage nodes store the triple indices as well as literal tables. The three triple indices are distributed onto the second group using a hash function to map the index keys from the triple component to a node identifier from the set of data nodes. Literal values can be resolved from NodeIDs, since this ID also includes the identifier of the node where the value is stored. Since every node can be located using the data present on the control nodes, any BGP with at least one fixed value can be resolved efficiently, with large possibilities of one-step query optimization.

While being able to deliver an impressive access performance, clustered TDB has several weaknesses: the system's query capability depends entirely on the mapping table on the query coordinators. Failures here affect the entire system, not only those nodes. Also, the issue of hot spots within the network has not been handled sufficiently: first, NodeIDs have the identifier of the storing node hard-coded inside them, which inhibits cheap reorganization. Second, the method to handle excessively used URIs (such as rdfs:label or rdf:type) with a replicated list of exceptions represents a global state, which is both likely to be inconsistent as well as dependent on manual interaction.

3.2 Garlik 4store

The clustered RDF store and SPARQL query system 4store were developed by the private company Garlik as an in-house storage system for several of their semantic web applications (Harris *et al.*, 2009). Since the applications to be supported were known, several shortcuts in system design were possible. 4store also uses a hash-based segmentation of the RDF data to be stored, and then stores these segments on a configurable number of storage nodes for replication. Rather than storing three indices, the distribution mechanism takes only the subject entry of the stored triples into account, which forces the system to flood queries for triples with an unknown subject entry to every node. As noted by the authors, this approach also fails for skewed data sets, where only few distinct subjects are used.

Similar to TDB, resources are not handled in their literal form, but rather mapped to an numerical representation using again a hash function, since numerical can be handled more efficiently. On each node, three data structures are kept to store RDF data: a hash map maps the numerical representation of resources back to their literal form. A prefix tree for each resource contains all predicates used with this resource, and a final table maps contains the stored named graphs. Query processing takes place

on the control nodes, with several optimizations reducing accesses to the storage nodes as well as several optimizations for join processing.

An interesting approach is used for the bootstrap process of the storage system. Rather than manually configuring the storage nodes and the segments stored on them on the control nodes, multicast DNS (Cheshire and Steinberg, 2005) is used to allow the control nodes to automatically discover all storage nodes and query them for the segments stored on them. This allows a comparably flexible and failure-tolerant operation for a clustered storage system.

3.3 OpenLink Virtuoso

Virtuoso is a commercial relational database management system developed by OpenLink Software, which has been adapted to also handle RDF data (Erling and Mikhailov, 2008). On a single machine, Virtuoso uses heavily compressed bitmap indices. For its clustered mode of operation, the database also uses hash partitioning to determine which node a operation is supposed to be executed on, with the mapping of partitions to nodes being manually created. To achieve failure-tolerance through redundancy, these partitions can also be stored and maintained on multiple nodes. Although it is not explicitly mentioned, it can safely be assumed that the mapping between partitions and nodes is stored on one or few "central" nodes. Should this be the case, it would represent crucial global state and also a single point of failure.

3.4 YARS2

As a part of a large and complex system for querying structured data, YARS2 also contains a storage component (Harth *et al.*, 2007). YARS2 has been designed to exhibit a linear scale-up with stored data volume. To accomplish this, six index structures designed to deliver results for different classes of triple patterns along with an inverted full-text index are distributed onto a fixed number of storage nodes. The mapping from index entries to nodes also uses a hash function. Since the hash-based distribution does not lead to an even distribution due to the small distinct number of keys, a random placement and query flooding were chosen for the predicate-index similar to 4store.

3.5 Summary

Clustered distributed storage systems for semantic data are a matured and commercially available technology. While their degree of self-organization is limited at best, their performance in the three dimensions we have identified can be satisfactory for many use cases. Scalability with regards to data size and query load can be achieved by distributing a significant portion of the stored data onto many nodes. All presented approaches are capable of this.

With their central control, clustered storage systems can adapt to skewed data shapes and unevenly distributed queries. The control nodes see all stored data items together with all queries and can thus optimize both their distribution scheme as well as query processing. However, only clustered TDB has acknowledged this issue by leaving its distribution scheme open. Out of practical reasons, YARS2 is most susceptible to skewed data, with its distribution decision only based on triple subjects. Robustness of the presented approaches is limited at best, the presence of distinguished nodes with crucial data structures jeopardizes the operation of the entire system, even though at least clustered TDB and 4store explicitly support multiple control nodes.

4. Self-organized storage

In an attempt to eliminate the last single point of failure in a clustered system – the control node – self-organized distributed systems work together to perform the tasks previously performed by the control node, effectively distributing control. In a self-organized system, no single node has any distinction to other nodes, all nodes share the same set of responsibilities. The main question to be answered in co-ordination is the lookup decision: where should a new data item be placed, and where does the system store the data items needed to answer a query. In a self-organized system, each node can act as a front-end to client applications, able to map queries to lookup operations, and then send these operations onto a path from node to node to their respective destinations. Results are then communicated back to the node where a request originated, which in turn sends them back to the client applications.

Self-organization has been defined as follows: "A system is *self-organized* if it is organized without any external or central dedicated control entity" (Prehofer and Bettstetter, 2005). We adopt this definition and the added general system characteristics of scalability, adaptability and robustness for distributed storage of semantic data; We expect satisfactory performance regarding scalability both to data and queries, adaptability to changing data shapes and query distributions and robustness toward node and link failures. Self-organized distributed storage systems are distinct from the much-researched peer-to-peer (P2P) systems. While employing similar techniques for distributed control, nodes in P2P systems are no longer under the control of one organization, while this is generally assumed for self-organized systems. Also, many P2P systems are focused only on the location of data items, and much less of the challenges inherent in also accepting data for storage.

In the following, we present different approaches on the fully distributed storage and retrieval of semantic data, and evaluate each approach to determine its degree of full self-organization. The tasks of fully distributed processing of complex queries as well as the organization of the persistent storage layer on each participating nodes are not in the scope of this review.

4.1 S-RDF

S-RDF is a fully distributed solution aimed solely at the discovery of RDF resources over a large unstructured network (Zhou *et al.*, 2009). The authors discard the possibility of using a structured P2P network due to the high impact the reorganization of overlay network structures has in quickly changing networks (Chawathe *et al.*, 2003). The approach chosen is the so-called biased multiple random walk (Lv *et al.*, 2002), where a configurable number of search requests is started on the node searching for data. Search requests are then routed from node to node using a list of known "neighbor" nodes advertised by an out-of-band mechanism, and where routing decisions are not based on a global law, but rather on independent decisions of the participating nodes. This routing decision is inherently random, thus retrieval operations might require an undue amount of hops inside the network before finding relevant data. To mend this issue, the randomness of the routing decision is biased according to meta data about RDF data stored on the neighboring nodes.

The RDF data to be published are analyzed by first finding all resources and their associated type(s). From these types, the most general type is selected and used to describe the particular set of RDF data. These types are then transmitted to

neighboring nodes, where they bias routing decisions for subjects of matching or derived types using an ontology-based similarity function. Furthermore, based on the percentage of times an incoming query was answered successfully, each node decides if a topology reorganization is needed. If so, another out-of-band mechanism is used to include peers storing data related to those locally stored into the neighbor list.

The approach to aggregate meta data and use this information in query routing relies on the presence of ontology information. In some cases, those can be extracted from the stored data, but only on the nodes actually storing the relevant data set. Other nodes are not able to determine the type structure without being given the ontology definition. Also, the mechanism to reorganize the network topology towards grouping peers hosting semantically related data sets together is also based on a globally consistent ontology structure, which may not exist. Also, generalization of defining types from RDF data may require manual interaction. An algorithm may be forced to summarize every file to the type owl:Thing, a class to which every RDF resource belongs to by definition.

4.2 SQPeer

The SQPeer concept is based on the distribution of schema information to efficiently route queries within the network (Kokkinidis and Christophides, 2004) and is similar to S-RDF. Each node that is part of a SQPeer network analyzes the schema information that is used within the locally stored RDF graphs. Classes and properties actually used in the local data are combined to the "active schema". This active schema is then distributed to every known node. Queries can now check if any of the known active schemata potentially contains data to fulfill the BGP within the query. The query is then forwarded to each node storing matching data after optimization.

The main issue with the SQPeer advertisement is the amount of active schema information that has to be exchanged between the nodes within the network. Since every peer can potentially use their own schema with an unlimited amount of classes and properties. Thus, the concept does not scale for heterogeneous data. Regarding the other extreme, where all data stored on all nodes is using the same schema, every query has to be forwarded to every node. Hence, the concept also does not scale for homogeneous data. Furthermore, queries can only include searching for nodes belonging to a particular class or annotated using a particular property. Other queries – for example requesting a single node from the various graphs – require flooding the request to all nodes.

4.3 Edutella

The Edutella system proposes an RDF-based P2P network infrastructure designed from a use case for improved networking between educational institutions (Nejdl *et al.*, 2002). Even though the name suggests a conceptual relationship to the unstructured Gnutella protocol, Edutella uses the JXTA framework developed by Sun Microsystems as its networking layer. JXTA in turn is based on the Tapestry concept of a distributed hash table (DHT) in the particular form of a prefix tree (trie) (Gong, 2001).

Tapestry provides de-centralized object location using a hash function to map stored objects and participating nodes to an opaque identifier (Zhao *et al.*, 2004). Each node is responsible for resolving a certain hash prefix based on its node identifier. In order to route requests, each node also keeps a list of neighboring nodes and their prefixes.

If a request for an unknown object hash arrives at a node, the request is forwarded to the neighbor node with the longest common prefix. Nodes publishing objects do so by resolving the node within the network responsible for the object's hash, and then store a pointer to themselves on that node. This way, if a request for a certain hash arrives at the responsible node, it can forward the request to the node where the object is actually stored. This scheme is shown in Figure 2: several nodes S1 through S6 are organized in a tree overlay network structure. Each node is responsible for a certain prefix, for example all key prefixed with 1.1 are routed to the node S3. Requests can be sent to any node, which will then use the overlay network to route for the longest matching prefix.

While being a very elegant object location scheme, the various indirections incur a high number of messages that have to be exchanged in order to locate an object. Also, if the distributed objects are not equally popular, some nodes in the network can be easily overloaded while other sit idle. Changes of the network structure always require the costly re-organization of node responsibilities in a considerable part of the network. Edutella uses the Tapestry model to allow peers to register the queries they have results for as Tapestry objects. Nodes trying to handle the user's queries can now find the nodes storing these objects, retrieve data from them, and thus answer the query. For example, a node could publish the fact that it stores RDF triples for a particular predicate URI, or even a predicate URI with a specific literal value. Edutella's development predates the standardization of the query language SPARQL, thus a custom query language family "RDF-QEL-i" with various levels of expressiveness is defined, however, due to very similar semantics, SPARQL support is feasible.

Even though Edutella completely abstracts the underlying network organization through its use of JXTA, it is still subject to the ability of the Tapestry concept to handle the unique properties of RDF data. While this can be beneficial for the design, if – for example – JXTA is extended to become more reliable (Xhafa *et al.*, 2008), it also implies the limitations of the underlying protocols onto Edutella's performance. As it has been noted previously, the RDF predicates rdfs:label or rdf:type are both occurring and requested frequently in many RDF data sets. If no further precautions are taken, every node that is part of an Edutella network would now publish the fact that it has data containing these predicates stored. The all-to common queries for these predicates would now result in a request to every node, which would inhibit scalability.

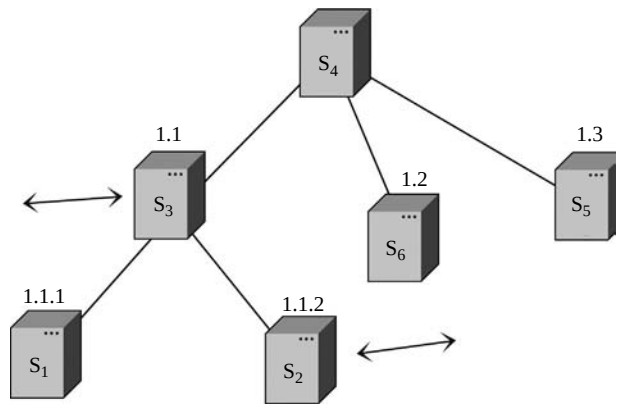

Figure 2.
Hash-trie network structure

As a refinement of the initial concept, Edutella has been extended to support two groups of nodes (Nejdl *et al.*, 2004). One group of "super nodes" acts as global query brokers with a complex overlay structure limiting the number of messages between them. A second group of simple "storage nodes" stores the actual data and is only communicating with super nodes. While certainly achieving the goals set by its authors, this network setup is no longer self-organized, and simply moves the problems described in the previous paragraph to another layer.

4.4 RDFPeers

RDFPeers is a P2P network of nodes storing RDF triples (Cai and Frank, 2004). In order to be able to resolve queries, a DHT is used to locate the nodes within the network responsible for storing a particular RDF triple. RDFPeers is based on the Chord network structure, where the overlay network is a modulo-ring (Stoica *et al.*, 2001). Here, each data item being stored is hashed using a globally known hash function, and each node is responsible for a particular range of the hash functions' result space. Each node also keeps pointers to the nodes responsible for the result space ranges following its own range – the so-called successor table – as well as pointers to other parts of the hash ring – the finger table – with exponentially increasing step size. Replication is performed by also storing new data items on a configurable number of nodes in the successor table. An overlay network using a ring structure is shown in Figure 3. Here, each node maintains a connection to at least one successor node in the ring. Requests sent to any node can be forwarded to the node responsible for the respective key range, for example a request for the key 1.1 sent to node S6 would be routed to S2 over S1.

Chord has been extended for the "Multi-Attribute Addressable Network for Grid Information Services" (Cai *et al.*, 2004), which is the direct basis of RDFPeers. In MAAN, three important changes are added to Chord: first, range queries become possible by using an order-preserving hash function. Second, the single-key limitation inherent in DHTs is extended to support multiple attributes for each stored data item. This is achieved by storing each item multiple times, each time with a different attribute as the DHT key. Query resolution is now performed by starting the query on one attribute

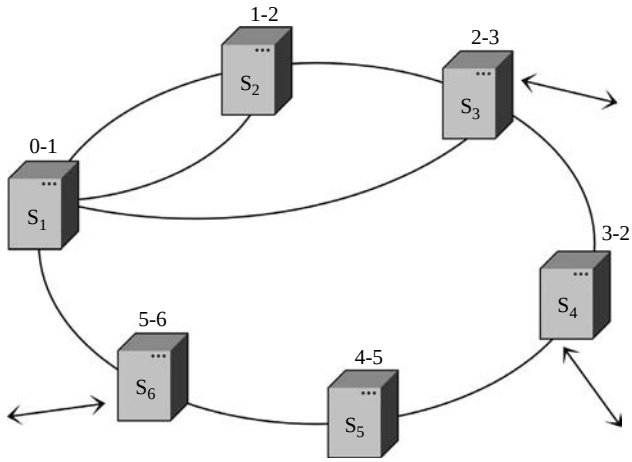

Figure 3.
Hash-ring network structure

while carrying the other restrictions as well, thereby achieving logarithmic performance on retrieval, with the storage performance growing linearly on the number of attributes. Third, load balancing, is achieved by not only using a locality preserving hash function, but a uniform locality preserving hash function. However, in order to calculate such a function, the statistical distribution function of the keys to be expected has to be known in advance.

Even though RDFPeers has been prudently designed to be scalable and to be able to answer almost all queries as well as store data with logarithmic effort with regards to the number of nodes to be visited, various issues are apparent: in order to create a uniform locality preserving hash function as required by MAAN to achieve an even storage load distribution, an analysis of the RDF graph to be stored is required. This manual interaction is undesirable. Skewed data popularity within queries is also not handled, and the hot spot issue for popular resources or predicates also applies (Battré *et al.*, 2006). Robustness is also seriously limited by relying on the ring overlay network structure, which requires costly reconstruction including movement should nodes fail or join the network.

4.5 GridVine

GridVine is a distributed semantic storage system built as an overlay over an arbitrary structured P2P network (Cudré-Mauroux *et al.*, 2007). Every node that is part of this P2P network participates in the storage of new data, thus increasing the capacity of the storage system becomes a trivial task. GridVine is built on top the P-Grid P2P system, which also uses a prefix tree similar to tapestry, but also includes several enhancements aimed towards self-organization: Within P-Grid, the prefix a node is responsible for is determined by a decentralized process determining if a change of responsibilities would be justified (Aberer *et al.*, 2003). This enables the P2P network to adapt to the actual distribution of the data object within the employed hash function's result space. The used overlay network structure is already shown in Figure 2.

Insertion of new triples into the distributed storage system can be performed on any node that is part of the P2P network. In order to enable other nodes to find triples by any of its entries, the object is added to the distributed index structure three times, once for each triple entry. Query operations can now translate triple patterns to P2P lookup operations, which statistically complete within logarithmic time. To achieve independence from node failures, a dynamic replication scheme samples the storage network to detect deviations from the desired replication level and then acts accordingly. Of course, if stored triples are changed, the replicated copies have to be located and updated as well, however, P-Grid supports a de-centralized mechanism based on rumor spreading for this task (Datta *et al.*, 2003).

GridVine also handles schemata, which describe the classes and properties used in the stored RDF graphs, these schemata can also be inserted into the storage layer, making them easily accessible to all nodes. Graphs using different schemata can be queried together, if a schema mapping has been defined. GridVine also supports the storage of these mappings, which can then be used on every node.

Through the usage of the advanced and self-organizing P-Grid infrastructure, GridVine avoids the issue of the uneven distribution of storage load, and thus scales very well with additional data to be stored as well as allows the system to adapt to changing properties of the stored data. The three-fold indexing requires no message

flooding for query purposes. However, the issue of uneven popularity of data items still poses an issue: for example, nodes unfortunate enough to be responsible for heavily used properties still can become hot points and thus hinder scalability.

4.6 The self-organized semantic storage service

The self-organized semantic storage service (S4) uses an entirely different approach for the organization of its distributed storage network: Here, the foraging and brood sorting methods used by several ant species is used to implement a distributed storage system (Mühleisen *et al.*, 2010, 2011; Tolksdorf and Augustin, 2009). Ant colonies are able to coordinate to millions of individuals without any central control, are able to adapt to changing environments, and can handle issues – e.g. ants being eaten – without affecting the rest of the colony. Foraging ants on the search for food start-off on a random walk from their nest, as soon as they encounter food, they return to the nest, leaving a pheromone path on the ground. Other ants on the search for food are now inclined – albeit not forced – to follow this path. Operations further increase pheromone values if they are successful as well. At any points, ants may also leave this path, enabling the discovery of new food sources. This algorithm has been shown to be able to create near-optimal solutions for hard combinatorial problems in affordable time. The adaption of these algorithms to distributed storage aims at also carrying these properties over, data items are modeled as the ant's food carrying a distinctive scent, and the storage network of interconnected nodes is their landscape. While S4 was designed to store tuples of arbitrary length, the storage of RDF triples is possible and was one of its use cases.

The overlay network structure of nodes connected to other nodes provides a virtual landscape, on which single storage and retrieval operations can move around. Operations are routed over multiple nodes according to distinct pheromone values present on the network connections, and successful operations trace back the path taken through the network, increasing the corresponding pheromone value. This represents a positive enforcement mechanism, converging on the shortest possible path from the node where a request was issued to the node storing the relevant data items over multiple consecutive requests. While it was shown that the described concept has a very high-success rate, a small percentage of requests will not reach their destinations, effectively accepting a trade-off between efficiency and correctness. This concept is shown in Figure 4: if a request to locate a data item #B stored on the node S3 is sent to node S2, this node can make a stochastic routing decision based on the pheromones present. In the example, the probability to route the request to S1 rather than to S5 is higher, since only one further hop will be required. On S1, the process is repeated, this time yielding an even higher probability. These values can be calculated at any node in the network, enabling routing all requests to their destination.

The second swarm concept used in S4 is the so-called "brood sorting", which is used by ants to organize their brood within their nest. Again, without any central organization, larvae in similar stages of development are placed near to each other, making caring for them a simpler task. The same concept is applied to the stored tuples in S4, according to an arbitrary similarity measure, tuples are clustered together into a limited number of clusters, making storage operations far more efficient.

Contrary to DHTs, the strictness of the global law governing the possible locations of data items is reduced, allowing the system for example to move data items in order to reach a even storage load distribution. The same property also allows the movement

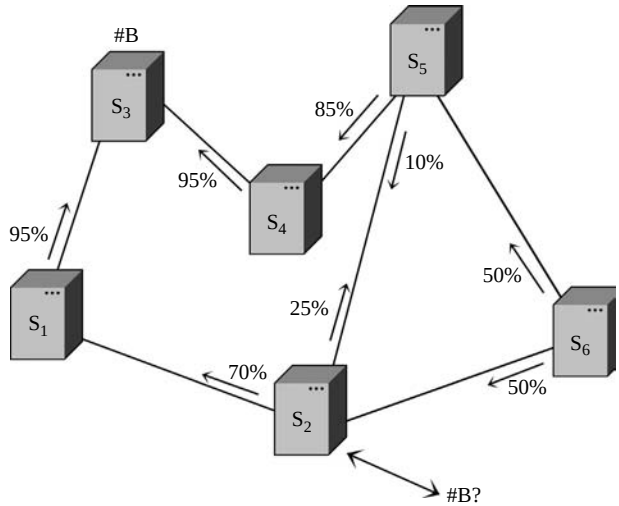

Figure 4.
S4 network structure

of over-popular data items to nodes with lesser load, making effective query load leveling possible. Should nodes fail or join the network, no re-organization of the stored data is necessary, since the swarm of queries will quickly form new paths around the failing nodes or include new nodes into the tasks of storing data and query answering.

4.7 Summary

Representing a logical next step from the clustered distributed storage systems presented in the preceding section, self-organizing storage systems do not only distribute the burden of storing the actual data items onto several nodes, but also the task of orchestrating the services of a distributed system onto the participating nodes themselves. Consistent with the literature, one can discern unstructured and structured self-organizing systems. The first two approaches S-RDF and SQPeer are using the unstructured paradigm with both flooding and random walk techniques. Edutella, RDFPeers and GridVine are using an structured approach with their DHTs. Finally, S4 uses an hybrid approach, where a emerging global data structure influences local decisions.

The unstructured approaches S-RDF and SQPeer showed sufficient scalability to the amount of data that can be discovered. However, both systems are unable to efficiently process BGP as a precursor to full SPARQL support, as their distribution only handles aggregates of the types used in the graphs stored on each node. Also, these aggregates are vulnerable to particular data distributions, which can seriously impede their adaptability. However, since no overlay network structures have to be maintained, robustness is high, node failures only affect the data published on them.

Edutella, RDFPeers and GridVine use DHT algorithms to achieve efficient query execution over multiple nodes. The properties of these algorithms lead to great scalability, which was their primary design goal. Their main drawback in the context of RDF storage and SPARQL query answering is their limited adaptability to skewed data and query distributions. Only GridVine with its underlying P-Grid network provides mechanisms that allow the system to adapt. In all cases, node or link failures

lead to costly network reconstruction, an issue becoming more severe as the storage network is growing.

The swarm-based S4 approach takes another trade-off between deterministic effort for query execution and the other dimensions. The algorithms based on swarm behavior are able to deliver unlimited scalability, robustness against failure as well as adaptability through the flexibility of the global laws.

5. Conclusion

While several interesting approaches for both clustered and self-organized storage of semantic information have been proposed, each approach chooses a different trade-off and thus faces its own limitations. To date, no "silver bullet" for the distributed storage of semantic information in the standardized form of RDF graphs has been proposed. Each solution can only deliver sufficient performance for very specific use cases. However, it is clear that clustered systems utilizing central nodes for the control of their services are only as robust as the control nodes. While this may be also acceptable in specific use cases, this is not an acceptable situation in the general case.

For self-organized systems, a general trade-off between storage load balance and query efficiency can be observed. In order to achieve query efficiency, the number of nodes potentially storing a requested data item has to be as low as possible, in the distributed setting this can only be achieved by using overlay network structures based on global laws – for example DHTs in their various forms. If the storage load is to be leveled out between nodes, queries cannot be routed as efficient anymore, instead random walks or directed flooding has to be used. As described for the respective approaches, the particular challenges in storing and querying RDF graphs can easily bring systems based on DHTs to their limits in adaptability due to wildly varying value frequencies. None of the mentioned systems is in commercial use, and the research prototypes that have been evaluated for publication were not available for a comparison. Hence, it appears to be still unclear whether RDF data can be distributed using DHT technologies in a generic and scalable fashion.

Table I shows an overview over the main properties of the concept presented in this survey. For each concept, method of distribution, its storage and retrieval capabilities, and degree of adaptability and robustness have been included as summarized in

Concept	Method	Storage	Retrieval	Adaptability	Robustness
Clustered					
Clustered TDB	Arbitrary	✓	✓	Limited	Limited
4store	Hash-based	✓	✓	–	Limited
Virtuoso	Hash-based	✓	✓	–	–
YARS2	Hash-based	✓	✓	–	–
Self-organized					
S-RDF	Random walk	–	✓	–	High
SQPeers	Schema flooding	–	✓	–	High
Edutella	DHT (tapestry)	–	✓	–	Limited
RDFPeers	DHT (chord)	✓	✓	–	Limited
GridVine	DHT (P-Grid)	✓	✓	High	Limited
S4	Swarms	✓	✓	High	High

Table I. Different types of semantic storage services and their properties

Sections 3.5 and 4.7. Since every approach was designed for scalability, this property has been omitted in the table.

None of the analyzed concepts was able to fulfill all requirements for large-scale storage and retrieval for semantic information. While being designed to scale to large amounts of data, their performance in correctly executing generic queries, handling skewed data sets, and efficiently reacting to changes in the network topology was only partly satisfying. Nonetheless, as each concept represented its own trade-off between these goals, we pointed out how self-organization is crucial to perform well at least in a subset of them. We therefore hope for further research to perform a practical evaluation of the concepts that were presented. As the semantic web lives up to its potential, these storage systems may be called for earlier than it can now be expected.

References

Aberer, K., Cudré-Mauroux, P., Datta, A., Despotovic, Z., Hauswirth, M., Punceva, M. and Schmidt, R. (2003), "P-grid: a self-organizing structured P2P system", *SIGMOD Rec.*, Vol. 32, September, pp. 29-33, available at: http://doi.acm.org/10.1145/945721.945729

Alpert, J. and Hajaj, N. (2008), "We knew the web was big…", available at: http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html

Battré, D., Heine, F., Höing, A. and Kao, O. (2006), "On triple dissemination, forward-chaining, and load balancing in DHT based RDF stores", in Moro, G., Bergamaschi, S., Joseph, S., Morin, J.-H. and Ouksel, A.M. (Eds), DBISP2P, Lecture Notes in Computer Science, Vol. 4125, Springer, Berlin, pp. 343-54, available at: http://dx.doi.org/10.1007/978-3-540-71661-7_33

Brickley, D. and Guha, R. (2004), *RDF Vocabulary Description Language 1.0: RDF Schema*, available at: www.w3.org/TR/rdf-schema/

Cai, M. and Frank, M. (2004), "RDFPeers: a scalable distributed RDF repository based on a structured peer-to-peer network", *WWW'04: Proceedings of the 13th International Conference on World Wide Web*, ACM Press, New York, NY, pp. 650-7, available at: http://doi.acm.org/10.1145/988672.988760

Cai, M., Frank, M.R., Chen, J. and Szekely, P.A. (2004), "MAAN: a multi-attribute addressable network for grid information services", *Journal of Grid Computing*, Vol. 2 No. 1, pp. 3-14, available at: www.springerlink.com/index/10.1007/s10723-004-1184-y

Chawathe, Y., Ratnasamy, S., Breslau, L., Lanham, N. and Shenker, S. (2003), "Making gnutella-like P2P systems scalable", in Feldmann, A., Zitterbart, M., Crowcroft, J. and Wetherall, D. (Eds), *Proceedings of the ACM SIGCOMM 2003 Conference*, ACM Press, New York, NY, pp. 407-18, available at: http://doi.acm.org/10.1145/863955.864000

Cheshire, S. and Steinberg, D.H. (2005), *Zero Configuration Networking – The Definitive Guide*, O'Reilly, Sebastopol, CA, available at: www.oreilly.de/catalog/bonjour/index.html

Cudré-Mauroux, P., Agarwal, S. and Aberer, K. (2007), "GridVine: an infrastructure for peer information management", *IEEE Internet Computing*, Vol. 11 No. 5, pp. 36-44.

Datta, A., Hauswirth, M. and Aberer, K. (2003), "Updates in highly unreliable, replicated peer-to-peer systems", *Proceedings of the 23rd International Conference on Distributed Computing Systems, ICDCS'03*, IEEE Computer Society, Washington, DC, pp. 76-85, available at: http://portal.acm.org/citation.cfm?id=850929.851977

DeWitt, D.J. and Gray, J. (1992), "Parallel database systems: the future of high performance database processing", Technical Report No. CS-TR-1992-1079, February, University of Wisconsin, Madison, WI.

Erling, O. and Mikhailov, I. (2008), "Towards web scale rdf", *Proceedings of the 4th International Workshop on Scalable Semantic Web Knowledge Base Systems* (*SSWS2008*), available at: http://data.semanticweb.org/workshop/ssws/2008/paper/main/1

Eubanks, N.D. (2005), "Bittorrent: digital river of the hacker culture", April 15, available at: http://hdl.handle.net/1901/189

Gong, L. (2001), "Industry report: JXTA: a network programming environment", *IEEE Internet Computing*, Vol. 5 No. 3, p. 88, available at: http://dlib.computer.org/ic/books/ic2001/pdf/w3088.pdf

Harris, S., Lamb, N. and Shadbol, N. (2009), "4store: the design and implementation of a clustered RDF store", in Fokoue, A., Guo, Y. and Liebig, T. (Eds), *The 5th International Workshop on Scalable Semantic Web Knowledge Base Systems* (*SSWS2009*), pp. 94-109, October, available at: http://ftp.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-517/ssws09-paper7.pdf

Harth, A., Umbrich, J., Hogan, A. and Decker, S. (2007), "YARS2: a federated repository for querying graph structured data from the web", *The Semantic Web, ISWC 2007, Busan, Korea, November 11-15*, Lecture Notes in Computer Science, Vol. 4825, Springer, Berlin, pp. 211-24.

Hayes, P. and McBride, B. (2004), "RDF semantics", available at: www.w3.org/TR/rdf-mt/

Klyne, G., Carroll, J.J. and McBride, B. (2004), "Resource description framework (RDF): concepts and abstract syntax", available at: www.w3.org/TR/rdf-concepts/

Kokkinidis, G. and Christophides, V. (2004), "Semantic query routing and processing in P2P database systems: the ICS-FORTH SQPeer middleware", in Lindner, W., Mesiti, M., Türker, C., Tzitzikas, Y. and Vakali, A. (Eds), *Proceedings of EDBT Workshops*, Lecture Notes in Computer Science, Vol. 3268, Springer, Berlin, pp. 486-95.

Lawrence, S. and Giles, C.L. (1999), "Accessibility of information on the web", *Nature*, Vol. 400, pp. 107-9.

Lv, Q., Cao, P., Cohen, E., Li, K. and Shenker, S. (2002), "Search and replication in unstructured peer-to-peer networks", in Leutenegger, S.T. (Ed.), *Proceedings of the 2002 International Conference on Measurement and Modeling of Computer Systems* (*SIGMETRICS-02*), *June 15-19*, volume 30, 1 of *SIGMETRICS Performance Evaluation Review*, ACM Press, New York, NY, pp. 258-9.

Mühleisen, H., Walther, T. and Tolksdorf, R. (2011), "Multi-level indexing in a distributed self-organized storage system", *IEEE Congress on Evolutionary Computation* (*CEC 2011*), IEEE Press, New Orleans, LA.

Mühleisen, H., Augustin, A., Walther, T., Harasic, M., Teymourian, K. and Tolksdorf, R. (2010), "A self-organized semantic storage service", *Proceedings of the 12th International Conference on Information Integration and Web-based Applications and Services, iiWAS'10*, ACM Press, New York, NY, pp. 357-64, available at: www.doi.acm.org/10.1145/1967486.1967542

Nejdl, W., Wolpers, M., Siberski, W., Schmitz, C., Schlosser, M., Brunkhorst, I. and Löser, A. (2004), "Super-peer-based routing strategies for RDF-based peer-to-peer networks", *Journal of Web Semantics*, Vol. 1 No. 2.

Nejdl, W., Wolf, B., Qu, C., Decker, S., Sintek, M., Naeve, A., Nilsson, M., Palmr, M. and Risch, T. (2002), "EDUTELLA: a P2P networking infrastructure based on RDF", *Proceedings of the Eleventh International World Wide Web Conference*, ACM Press, New York, NY.

Owens, A., Seaborne, A., Gibbins, N. and Schraefel, M.C. (2008), "Clustered TDB: a clustered triple store for Jena", Technical Report, Electronics and Computer Science, University of Southampton, available at: http://eprints.ecs.soton.ac.uk/16974/

Prehofer, C. and Bettstetter, C. (2005), "Self-organization in communication networks: principles and design paradigms", *IEEE Communications Magazine*, Vol. 43 No. 7, pp. 78-85, available at: http://dx.doi.org/10.1109/MCOM.2005.1470824

Prud'hommeaux, E. and Seaborne, A. (2008), "SPARQL query language for RDF", available at: www.w3.org/TR/rdf-sparql-query/

Sakr, S. and Al-Naymat, G. (2010), "Graph indexing and querying: a review", *IJWIS*, Vol. 6 No. 2, pp. 101-20.

Stoica, I., Morris, R., Karger, D.R., Kaashoek, M.F. and Balakrishnan, H. (2001), "Chord: a scalable peer-to-peer lookup service for internet applications", *Proceedings of the ACM SIGCOMM, San Diego, CA, USA*, pp. 149-60, available at: http://doi.acm.org/10.1145/383059.383071

Tolksdorf, R. and Augustin, A. (2009), "Selforganisation in a storage for semantic information", *Journal of Software*, Vol. 4 No. 8, pp. 798-807.

Xhafa, F., Barolli, L., Fernandez, R., Daradoumis, T. and Caballé, S. (2008), "Extension and evaluation of JXTA protocols for supporting reliable p2p distributed computing", *IJWIS*, Vol. 4 No. 1, pp. 121-35.

Zhao, B.Y., Huang, L., Stribling, J., Rhea, S.C., Joseph, A.D. and Kubiatowicz, J. (2004), "Tapestry: a resilient global-scale overlay for service deployment", *IEEE Journal on Selected Areas in Communications*, Vol. 22 No. 1, pp. 41-53, available at: http://doi.ieeecomputersociety.org/10.1109/JSAC.2003.818784

Zhou, J., Hall, W. and Roure, D.D. (2009), "Building a distributed infrastructure for scalable triple stores", *J. Comput. Sci. Technol.*, Vol. 24 No. 3, pp. 447-62, available at: http://dx.doi.org/10.1007/s11390-009-9236-1

About the authors

Hannes Mühleisen studied Software Engineering at the Universität Stuttgart and Computer Science at the Humboldt-Universität zu Berlin. He graduated in 2010 with a thesis on Access Policies for Linked Data. Currently pursuing a PhD in distributed systems as a Postgraduate Researcher at the Freie Universität Berlin, focusing on using swarm intelligence for distributed storage systems. Hannes Mühleisen is the corresponding author and can be contacted at: muehleis@inf.fu-berlin.de

Tilman Walther graduated in 2008 from Freie Universität Berlin with a Diploma in Computer Science and is currently working as Postgraduate Research Associate with focus on swarm intelligence and network science at the group on networked information systems.

Robert Tolksdorf graduated in Computer Science at Technische Universität Berlin and was appointed Professor at Freie Universität Berlin in 2002. His areas of work and interest are semantic technologies, self-organization for coordination and network analysis.