A Survey of Structured P2P Systems for RDF Data Storage and Retrieval*

Imen Filali, Francesco Bongiovanni, Fabrice Huet, and Françoise Baude

INRIA Sophia Antipolis
CNRS, I3S, University of Nice Sophia Antipolis
France
first.last@inria.fr

Abstract. The Semantic Web enables the possibility to model, create and query resources found on the Web. Enabling the full potential of its technologies at the Internet level requires infrastructures that can cope with scalability challenges while supporting expressive queries. The attractive features of the Peer-to-Peer (P2P) communication model, and more specifically *structured P2P systems*, such as decentralization, scalability, fault-tolerance seems to be a natural solution to deal with these challenges. Consequently, the combination of the Semantic Web and the P2P model can be a highly innovative attempt to harness the strengths of both technologies and come up with a scalable infrastructure for RDF data storage and retrieval. In this respect, this survey details the research works adopting this combination and gives an insight on how to deal with the RDF data at the indexing and querying levels. We also present some works which adopt the publish/subscribe paradigm for processing RDF data in order to offer long standing queries.

Keywords: Semantic Web, Peer-to-Peer (P2P), Distributed Hash Tables (DHTs), Resource Description Framework (RDF), Distributed RDF repository, RDF data indexing, RDF query processing, publish/subscribe (pub/sub), subscription processing.

1 Introduction

The realization of the Semantic Web [7] and more specifically the "Linked Data" vision [18] of Sir Tim Berners Lee at the Internet level has seen the emergence of a new breed of distributed systems that combines Peer-to-Peer (P2P) technologies with the Resource Description Framework (RDF) (metadata) data model [4]. This combination, which allows a flexible description and sharing of both data and metadata in large scale settings, arose quite naturally, and, as stated in [82], both address the same need but at different levels. The Semantic Web allows

* The presented work is funded by the EU FP7 NESSI strategic integrated project SOA4All (http://www.soa4all.eu), EU FP7 STREP project PLAY (http://www.play-project.eu) and French ANR project SocEDA (http://www.soceda.org)

users to model, manipulate and query knowledge at a conceptual level with the intent to exchange it. P2P technologies, on the other hand, enable users to share the information, encompassing the need of its indexing and storage, using a decentralized organization. P2P systems have been recognized as a key communication model to build scalable platforms for distributed applications such as file sharing (e.g., Bittorrent [29]) or distributed computing (e.g., SETI@home [5]). P2P architectures are classified into three main categories: *unstructured*, *hybrid* and *structured* overlays. In *unstructured* P2P overlays, there is no constraint on the P2P architecture as the overlay is built by establishing random links among nodes [1]. Despite their simplicity, they suffer from several issues: limited scalability, longer search time, higher maintenance costs, etc. A *hybrid* P2P overlay is an architecture in which some parts are built using random links and some others, generally the core of the overlay, are constructed using a specific topology [87]. *Structured Overlay Networks* (SONs) [36] emerged to alleviate inherent problems of unstructured overlays. In these systems, peers are organized in a well-defined geometric topology (e.g., a ring, a torus, etc.) and, compared to unstructured networks, they exhibit stronger guarantees in terms of search time and nodes' maintenance [27]. By providing a *Distributed Hash Table (DHT)* abstraction, by the means of *Put* and *Get* methods, they offer simple mechanisms to easily store and fetch the data in a distributed environment. However, even if this abstraction is practical, it has its limits when it comes to *efficiently* support the types of queries that the Semantic Web technologies intend to provide. The main goal behind these systems is a simple *data storage* and *retrieval*. Therefore, a key point in building a scalable distributed system is how to efficiently index data among peers participating in the P2P network in order to ensure efficient lookup services, and thus improve the efficiency of applications and services executed at the top level.

The question we try to address in this survey is the following: how to store the RDF data and evaluate complex queries expressed in various Semantic Web query languages (e.g., [6, 50, 81]) on top of fully distributed environments? In this regard, we survey several approaches for P2P distributed storage systems that adopt RDF as a data model. In particular, we focus on data indexing, data lookup and query evaluation mechanisms. We also give an overview of the publish/subscribe (pub/sub) communication paradigm build on top of P2P systems for RDF data storage and retrieval. Rather than pursuing an exhaustive list of P2P systems for RDF data storage and retrieval, our aim is to provide a unified view of algorithmic approaches in terms of data indexing and retrieval, so that different systems can be put in perspective, compared and evaluated. Several survey papers focusing on P2P networks (e.g., [51, 62, 68, 77]) or the pub/sub communication model (e.g., [38, 60]) have been proposed. None of them addressed the combination of the P2P architecture and the RDF data model. The work presented by Lua *et al.* in [62] gives an extensive comparative study of the basic structured and unstructured P2P systems which is out of the scope of this paper. Related surveys have been presented in [68, 77] but even if they have a wider scope than [62], as they cover several service discovery frameworks, search methods, etc., they do not detail the proposed mechanisms. This survey, on the contrary, presents an in depth analysis of carefully selected papers that combine the RDF data model and the P2P technologies.

In order to be self-contained, Section 2 introduces the context of this paper and gives an insight of the basic terminologies and technologies that will be considered while investigating RDF-based P2P systems. This includes the main concepts behind the RDF data model as well as a taxonomy of P2P systems from the overlay architecture point of view. Section 3 discusses several P2P systems that combine a P2P architecture with the RDF data model. Specifically, it focuses on the data indexing and query processing mechanisms and also investigates the pub/sub paradigm on top of RDF-based P2P systems. Moreover, it underlines various algorithms that have been proposed to manage complex subscriptions. Summary and general discussions are laid out in Section 4 before concluding the survey in Section 5.

2 Context and Background

The main idea behind the Semantic Web is to define and link the Web content in a machine understandable way. Realizing this vision requires well defined knowledge representation models and languages to create, model and query resources on the Web. Several technologies have been proposed in this direction to accomplish this goal at the conceptual level [7]. The processing of a huge amount of information at the Web scale, from an architectural point of view, is not a trivial task and the ever-growing increase of the resources available on the Web is overwhelming for traditional centralized systems. Client/server-based solutions suffer from scalability issues so there is a need for a (fully) distributed solution to cope with this problem. After the success that has been revealed by the P2P communication model, the idea of exploiting its strengths (e.g., distributed processing, fault-tolerance, scalability, etc.) to answer the requirements of the Semantic Web layers, and more specifically the RDF data model, has been explored. Besides the attempt to combine the strengths of the RDF and the P2P models, the need for *up-to-date* data requires more advanced algorithms and techniques in order to immediately retrieve data when it gets stored in the system. The publish/subscribe communication paradigm seems to be a natural solution to deal with this issue. In this respect, we will investigate along the remainder of this paper to which extent the Semantic Web can benefit from the P2P community in accordance with RDF data storage and retrieval.

Before presenting and discussing a set of selected works,we give in the following section the basic concepts behind the RDF model. Then, we briefly describe the main overlay architectures as well as the key ideas behind the pub/sub paradigm. Finally, we outline some challenges related to RDF data and query processing on top of P2P systems. Note that although several technologies such as the Resource Description Framework Schema (RDFS), the Web Ontology Language (OWL) are among the main building blocks of Semantic Web stack, we limit ourselves to RDF data storage and retrieval and related query mechanisms on top of P2P systems.

2.1 The RDF Data Model

The Resource Description Framework (RDF) [4] is a W3C standard aiming to improve the World Wide Web with machine processable semantic data. RDF provides a powerful abstract data model for knowledge representation which can be serialized in XML or Notation3 (N3) formats. It has emerged as the prevalent data model for the Semantic Web [7] and is used to describe semantic relationships among data. The notion of RDF *triple* is the basic building block of the RDF model. It consists of a *subject (s)*, a *predicate (p)* and an *object (o)*. More precisely, given a set of *URI* references $\mathcal{R}$, a set of blank nodes $\mathcal{B}$, and a set of literals $\mathcal{L}$, a triple $(\mathtt{s}, \mathtt{p}, \mathtt{o}) \in (\mathcal{R} \cup \mathcal{B}) \times \mathcal{R} \times (\mathcal{R} \cup \mathcal{B} \cup \mathcal{L})$ The subject of a triple denotes the *resource* that the statement is about, the predicate denotes a *property* or a characteristic of the subject, and the object presents the *value* of the property. Note that there is another way to look for the RDF information which is a labeled directed connected graph. Nodes are the subjects and the objects of the RDF statements, while edges refer to the predicates of the triples. Edges are always directed from the subjects to the objects. This presentation gives an easy to understand visual explanation of the relationship between the RDF triples elements. Figure 1 shows an example of a RDF graph.

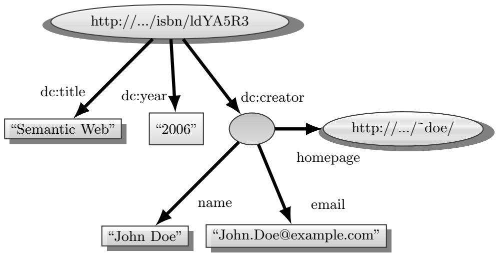


Fig. 1. An Example of a RDF graph. Plain nodes literals are presented with rectangles. Nodes with URI references are presented by oval shapes. Empty gray node presents a blank node.

RDFS [32] is an extension of RDF providing mechanisms for describing groups of resources and their relationships. Among other capabilities, it allows to define classes of resources and predicates. Moreover, it enables to specify constraints on the subject or the object of a given class of predicates.

In addition to the definition of the RDF data model which will be considered in our further discussions, we need to define and classify different types of queries considered in the presented works:

– **Atomic queries** are triple patterns where the subject, predicate and object can either be variables or constant values. According to [23], atomic triple queries can be summarized into eight queries patterns: $(?\mathtt{s}, ?\mathtt{p}, ?\mathtt{o})$, $(?\mathtt{s}, ?\mathtt{p}, \mathtt{o_i})$,$(?\mathtt{s}, \mathtt{p_i}, ?\mathtt{o})$, $(?\mathtt{s}, \mathtt{p_i}, \mathtt{o_i})$, $(\mathtt{s_i}, ?\mathtt{p}, ?\mathtt{o})$, $(\mathtt{s_i}, ?\mathtt{p}, \mathtt{o_i})$, $(\mathtt{s_i}, \mathtt{p_i}, ?\mathtt{o})$, $(\mathtt{s_i}, \mathtt{p_i}, \mathtt{o_i})$. In each query pattern, $?\mathtt{x}$ denotes the element(s) that the query is looking for and $\mathtt{x_i}$ refers to a constant value of the RDF element. For instance, the query pattern $(?\mathtt{s}, \mathtt{p_i}, \mathtt{o_i})$ returns all subjects $\mathtt{s}$ for a given predicate $\mathtt{p_i}$ and object $\mathtt{o_i}$ (if they exist).

– **Conjunctive queries** are expressed as a conjunction of a list of atomic triple patterns (sub-queries). For instance, a conjunctive query may look like: $(?\mathtt{s_1}, \mathtt{p_1}, \mathtt{o_1}) \wedge (?\mathtt{s_2}, \mathtt{p_2}, \mathtt{o_2})$.
– **Range queries** involve queries for single or multiple attribute data whose attribute value falls into a range defined by the query. For instance $\mathtt{q} = (\mathtt{s}, \mathtt{p}, ?\mathtt{o})$ FILTER $\mathtt{o} < \mathtt{v}$, looks, for a given subject $\mathtt{s}$ and a predicate $\mathtt{p}$, for all object values bounded by the value $\mathtt{v}$.
– **Continuous queries** are a particular type of queries which are decoupled in time, space and synchronization between peers participating in the query resolution. This kind of querying is found in pub/sub systems where subscribers of a given query can continuously receive matching results in the form of asynchronous notifications. Any type of queries (atomic, conjunctive, etc.) can be used in an asynchronous manner.

2.2 P2P Architectures

Unstructured P2P overlays. In this model, peers form a loosely coupled overlay without any constraint neither on the network topology nor on the data placement. Peers are inherently flexible in their neighbor selection, that is, when a new peer joins the P2P network, it freely selects its neighbors to connect to. The resulting graph characteristics, such as peers' degree, depend on the way peers are connected. Therefore, connections between peers may be based on purely randomized approaches or it can benefit,for instance, from other aspects such as semantic clustering [30]. Despite the unstructured P2P architectures are appealing due to their properties of fault-tolerance, these overlays often provide restricted guarantees to locate the data even if it is available in the network as the life time of the query is generally limited by a fixed number of hops. Techniques based on *gossip-based* algorithms [34, 37], inspired by the way infections diseases spread, offer solutions for improving search in unstructured overlays.

Hybrid P2P overlays. In this model, a set of peers act as super-peer nodes for a set of "ordinary" peers and can perform more complex tasks such as data indexing and query routing (e.g., [61, 72, 87]). Super-peers could be connected to each other in a structured manner while the rest of the nodes establishes random connections with one another.

Structured P2P overlays. Unlike unstructured P2P systems, structured P2P overlays, are built according to a well-defined geometric structure as well as deterministic links between nodes. Due to these properties, they can offer guarantees regarding the lookup time, scalability and can also decrease the maintenance cost of the overlay [27]. P2P overlays such as CAN [75], Chord [83], Pastry [78], P-Grid [8] and many others present an attractive solution for building large scale distributed applications thanks to the practical DHT abstraction that they offer. Although they differ in several design choices such as the network topology, the routing mechanisms, they provide a *lookup* function on top of which they offer this DHT abstraction. Most of DHT-based systems share a common idea, that is, the mapping of a key space to a set of peers such that each peer is responsible for a partition of that space. The *Put* and *Get* methods provided by the DHT abstraction are respectively responsible for storing and retrieving the data. The use of hashing functions by DHTs guarantees, with high probability, a uniform distribution of the data. This valuable property correlated with all previous ones make DHTs the most suitable choice as a P2P substrate.

2.3 Publish/Subscribe in P2P Systems

In this section, we briefly highlight the main concepts underlying the pub/sub communication model. We then focus on the synergy between this particular paradigm and RDF-based P2P systems, in order to see how complementary a dynamic messaging paradigm and a rich data model could be. By going into details of the subscription and advertisement processing algorithms, we will present some ground works that were done around this promising symbiosis.

In a pub/sub system, subscribers, also called consumers, can express their interests in an event or a pattern of events, and be notified of any generated event by the publishers (producers) that matches those interests. The events are propagated asynchronously to all subscribers. Therefore, the overall system is responsible for matching the events to the subscriptions and for the delivery of those relevant events from the publishers to the subscribers which are distributed across a wide area network. This paradigm provides a decoupling in time, space and synchronization between participants. First, subscribers and publishers do not need to participate in the relation at the same time. Secondly, senders and receivers are not required to have a prior knowledge of each other and can even be located in separate domains. Finally, they are not blocked when generating events and subscribers can get the notifications in an asynchronous manner. Overall, the key components of a pub/sub system can be summarized into the following concepts:

– **Subscriptions.** A subscription describes a set of notifications a consumer is interested in. The goal behind the subscription process is that subscribers will receive notifications matching their interests from other peers in the network. Subscriptions are basically *filters*, which can range from simple boolean-valued functions to the use of a complex query language. The expressiveness of the subscriptions in terms of filtering capabilities depends directly from the data and the filter models used. Note that when discussing the pub/sub model, the words "subscription" and "query" are interchangeable.
– **Notifications.** In pub/sub systems, notifications can signify various types of events depending on the perspective. From a publisher perspective, *advertisements* are a type of notification used to describe the kind of notification the publishers are willing to send. From a subscriber perspective, a *notification* is an event that matches the subscription(s) of consumers which is responsible for conveying notifications to subscribers. Several peers within

the network can actively or passively participate in the dissemination of those notifications.

Pub/sub systems have several ways for identifying notifications that can be based either on a *topic* or the *content*. In the *topic-based* model, publishers annotate every event they generate with a string denoting a distinct topic. Generally, a topic is expressed as a rooted path in a tree of subjects. For instance, an online research literature database application (e.g., Springer) could publish notifications of newly available articles from the *Semantic Web* research area under the subject "/Articles/Computer Science/Proceedings/Web/Semantic Web". This kind of topic will then be used by subscribers which will, upon subscription's generation, explicitly specify the topic they are interested in and for which they will receive every related notifications. The topic-based model is at the core of several systems such as Scribe [26], Sub-to-Sub [86]. A main limitation of this model lies in the fact that a subscriber could be interested only in a subset of events related to a given topic instead of all events. This comes from the tree-based classification which severely constrains the expressiveness of the model as it restricts notifications to be organized using a single path in the tree. A tree-based topic hierarchy inhibits the usage of multiple super-topics for instance, even if some inner re-organizations are possible, this classification mechanism remains too rigid. On the other side, a *content-based* model is much more expressive since it allows the evaluation of filters on the whole content of the notifications. In other words, it is the data model and the applied predicates that exclusively determine the expressiveness of the filters. Subscribers express their interests by specifying predicates over the content of notifications they want to receive. These constraints can be more or less complex depending on the query types and operators that are offered by the system. Available query predicates range from simple comparisons, regular expressions, to conjunctions, disjunctions, and XPath expressions on XML. As the focus of this survey is RDF data storage and retrieval on top of P2P networks, we only consider pub/sub systems which solely address such combination. Therefore, this overview complements other surveys related to the pub/sub paradigm. The interested reader can refer to [38, 60, 70] for deeper discussions on the basic concepts behind this communication model.

2.4 RDF Data Processing on Top of P2P Systems: What Are the Main Challenging Aspects ?

The P2P communication model has drawn much attention and has emerged as a powerful architecture to build large scale distributed applications. Earlier research efforts on P2P systems have focused on improving the scalability of unstructured P2P systems by introducing specific geometries for the overlay network, i.e., SONs and adding a standard abstraction on top of it, i.e., DHT. The first wave of SONs [75, 83] focused on scalability, routing performances and churn resilience. The main issue of these DHTs is that they only support *exact queries*, i.e., querying for data items matching a given key (`lookup(key)`). To overcome such limitation, a second wave of SONs led to a set of P2P approaches such as [8, 22] having the capabilities to manage more complex queries such as range queries [8] or multi-attribute queries [22]. However, storage and retrieval of RDF data in a distributed P2P environment raise new challenges. In essence, as the RDF data model supports more complex queries such as conjunctive and disjunctive queries, an additional effort to design adequate algorithms and techniques to support advanced querying beyond simple keyword-based retrieval and range queries is required. Consequently, a particular attention has to be made regarding the data indexing since it has a great impact on the query processing phase. Overall, two main aspects have to be taken into consideration while investigating RDF data storage and retrieval in P2P systems:

Data indexing. How can we take advantage of the format of the RDF data model in order to efficiently index RDF triples?

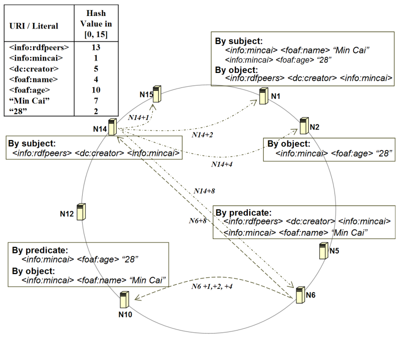
Query processing. How can we take advantage of P2P search mechanisms to efficiently query and retrieve RDF data in large scale settings? Moreover, whenever a query, composed of a set of sub-queries, can be answered by several nodes, how to efficiently combine RDF data residing in different locations before sending the result back to the user? In addition, what is the impact of the data indexing methods on the query processing algorithms?

3 RDF Data Storage and Retrieval in P2P Systems

Centralized RDF repositories and lookup systems such as RDFStore [3], Jena [2], RDFDB [79] have been designed to support RDF data storage and retrieval. Although these systems are simple in design, they suffer from the traditional limitations of centralized approaches. As an alternative to these centralized systems, P2P approaches have been proposed to overcome some of these limitations by building (fully) decentralized data storage and retrieval systems. The remainder of this section presents a selection of research works combining the RDF data model with a P2P model with a focus on data indexing and query processing. Note that we do not consider in our discussions unstructured P2P approaches for RDF data storage and retrieval such as Bibster [46] and S-RDF [88], that we have discussed in [39] and we only overview P2P systems based on structured overlays. These systems differ in the overlay architecture, data indexing and query processing mechanisms. These approaches, presented in the next sections, are classified according to the overlay structure (ring-based, cube-based, tree-based) in case they are directly related to the underlying structure. Otherwise, they are grouped into a class called “generic DHT-based”. In Section 3.5, we present a set of pub/sub systems dealing with asynchronous queries. Therefore, this overview complements other surveys related to the pub/sub paradigm. The interested reader can refer to [38, 60, 70] for deeper discussions on the basic concepts behind this communication model. Note that along the remainder of this work, we use the original notation found in the presented papers.

3.1 Ring-Based Approaches

RDFPeers. RDFPeers [23] was the first P2P system that has proposed the use of DHTs in order to implement a distributed RDF repository. It is built



URI / Literal	Hash Value in [0, 15]
<info:rdfpeers>	13
<info:mincai>	1
<dc:creator>	5
<foaf:name>	4
<foaf:age>	10
"Min Cai"	7
"28"	2

By subject:
<info:mincai> <foaf:name> "Min Cai"
<info:mincai> <foaf:age> "28"
By object:
<info:rdfpeers> <dc:creator> <info:mincai>
N15
N1
N14+1
N14
N2
N14+2
By subject:
<info:rdfpeers> <dc:creator> <info:mincai>
By object:
<info:mincai> <foaf:age> "28"
N14+4
N14+8
N12
N6+8
By predicate:
<info:rdfpeers> <dc:creator> <info:mincai>
<info:mincai> <foaf:name> "Min Cai"
N5
By predicate:
<info:mincai> <foaf:age> "28"
By object:
<info:mincai> <foaf:name> "Min Cai"
N6 +1,+2, +4
N6
N10

Fig. 2. Example of RDF data storage in RDFPeers network with 8 present nodes in a 4 bit identifier space (taken from[23])

on top of MAAN [22](Multiple Attribute Addressable Network) which is an extension of Chord [83]. As in Chord, each node has an identifier that represents its position in a circular identifier space of size N, and has direct references to its predecessor and successor in the ring. Moreover, a node keeps $\mathtt{m} = \mathtt{log}(\mathtt{N})$ routing entries, called *fingers*. A `finger[i]` equals to $\mathtt{successor}(\mathtt{n} \oplus 2^{\mathtt{i}-1})$ with $1 \leq \mathtt{i} \leq \mathtt{m}$, where $(\mathtt{n} \oplus 2^{\mathtt{i}-1})$ is $(\mathtt{n} + 2^{\mathtt{i}-1})$ `modulo N`. A `successor` of a node `n` is defined as the first node that succeeds `n` along the clockwise direction in the ring space. Figure 2 shows a RDFPeers network with `8` present nodes in a `4` bit identifier space. It also depicts the fingers of nodes $\mathtt{N}_6$ and $\mathtt{N}_{14}$ presented by dashed arrows originated respectively from $\mathtt{N}_6$ and $\mathtt{N}_{14}$. Data indexing and query processing mechanisms in RDFPeers work as follows:

Data indexing. RDFPeers builds a distributed index. More precisely, a RDF triple, labeled `t=(s,p,o)`, is indexed three times by applying a hash function on the subject (`hash(s)`), the predicate (`hash(p)`) and the object (`hash(o)`). The triple `t` will then be stored on peers responsible for those hashed values. For instance, to store a triple `t` according to the predicate element, the node which is responsible for `hash(p)` will store the triple `t`. For better understanding consider the storage of the following three triples in the RDFPeers infrastructure depicted in Figure 2: `<info:rdfpeers><dc:creator><info:mincai>`; `<info:mincai><foaf:name>"Min Cai"`; `<info:mincai> <foaf:age> "28" ^^ <xmls:integer>`. Figure 2 depicts the hashed values of all the URI and literals values of these triples. Suppose now that node $\mathtt{N}_6$ receives a `store` message aiming to insert the triple `<info:mincai> <foaf:age> "28" ^^<xmls:integer>` in the network according the hashing of the predicate element (i.e., `<foaf:age>`). As `hash(<foaf:age>)=10` (see table in Figure 2), $\mathtt{N}_6$, and based on its fingers, will route the query to the $\mathtt{N}_{10}$.

Query processing. RDFPeers supports three kinds of queries.

- **Atomic triple pattern query.** In this kind of query, the subject, the predicate, or the object can either be a variable or an exact value. For instance, a query like `(s,?p,?o)` will be forwarded to the node responsible for `hash(s)`. All atomic queries take $\mathcal{O}(\mathtt{log}(\mathtt{N}))$ routing hops to be resolved except queries in the form of `(?s,?p,?o)` which require $\mathcal{O}(\mathtt{N})$ hops in a network of `N` peers.

– **Disjunctive and range query.** RDFPeers optimizes this kind of query through the use of the locality preserving hash function. Indeed, when the variable's domain is limited to a range, the query routing process starts from the node responsible for the lower bound. It is then forwarded linearly until received by the peer responsible for the upper bound. In the case of disjunctive range query like $(\mathtt{s}, \mathtt{p}, ?\mathtt{o}), ?\mathtt{o} \in \cup_{\mathtt{i}=1}^{\mathtt{n}}[\mathtt{l_i}, \mathtt{u_i}]$, such as $\mathtt{l_i}$ and $\mathtt{u_i}$ are respectively the lower and the upper bound of the range `i`, several ranges have to be satisfied, intervals are sorted in ascending order. The query will then be forwarded from one node to the other, until it will be received by the peer responsible for the upper bound of the last range. Disjunctive exact queries such as $(\mathtt{s}, \mathtt{p}, ?\mathtt{o}), \mathtt{o}? \in \{\mathtt{v_1}, \mathtt{v_2}\}$ are resolved using the previous algorithm since they are considered as a special case of disjunctive range queries where the lower and the upper bounds are equal to the exact match value.
– **Conjunctive query.** RDFPeers supports this type of queries, as long as they are expressed as a conjunction of atomic triples patterns or disjunctive range queries for the *same* subject. Constraints' list can be related to predicates or/and objects. To resolve this type of query, authors use a *multi-predicate query resolution* algorithm. This algorithm starts by recursively looking for all candidate subjects on each predicate and intersects them at each step before sending back the final results to the query originator. More precisely, let us denote by `q` the current active sub-query; $\mathcal{R}$ the list of remaining sub-queries that have to be resolved; $\mathcal{C}$ the list of candidate subjects that matches the current active sub-query `q` and $\mathcal{I}$ the set of intersected subject that matches *all already resolved* sub-queries. Whenever subjects that match the sub-query `q` are found, they will be added to $\mathcal{C}$. At each forwarding step, an intersection between $\mathcal{C}$ and $\mathcal{I}$ is made in order to keep only subjects that match all already resolved sub-queries. Once *all* results for the current active query are found,the first sub-query in $\mathcal{R}$ is popped to `q`. Whenever all sub-queries are resolved (i.e., $\mathcal{R} = \{\emptyset\}$) or no results for the current sub-query are reported (i.e., $\mathcal{I} = \{\emptyset\}$), the set $\mathcal{I}$ containing the query results will be sent back to the originator.

While it supports several kinds of queries, RDFPeers has a set of limitations especially during the query resolution phase. This includes the *attribute selectivity* and the restrictions made at the level of supported queries. The attribute selectivity is associated with the choice of the first triple pattern to be resolved. Low selectivity of an attribute leads to a longer computational time to manage the local search as well as a greater bandwidth consumption to fetch results from one node to the other, because many triples will satisfy that constraint. As an example, the predicate `'rdf:type'` seems to be less selective, as it can be more frequently used in RDF triples than others. Despite the attribute selectivity parameter having an important impact on the performance of the query resolution algorithm, RDFPeers does not provide a way to estimate such parameter. The pattern selectivity is also considered in [15] where lookups by subject have priority over lookups by predicates and objects and lookups by objects have priority over lookup by predicates. Another issue is related to conjunctive triple pattern queries which are not fully supported and are restricted to conjunctive queries with the same subject so that arbitrary joins are not supported.

Atlas. A similar approach to RDFPeers was presented in the Atlas project [54] which is a P2P system for RDFS data storage and retrieval. It is built on top of Bamboo DHT [76] and uses the RDFPeers query algorithm. Atlas uses RDF Query Language (RQL) [50] as query language and the RDF Update Language (RUL) [28] for RDF metadata insertion, deleting and update.

Subscription processing. Atlas supports a content-based pub/sub paradigm, using algorithms from [59]. In essence, after the submission of an atomic query and its indexing, a peer will wait for triples that satisfy it. Once a new triple is inserted, nodes cooperate together in order to determine which queries are satisfied. Nodes responsible for triples satisfying the subscription create notifications and send them to the subscriber node. Processing of conjunctive subscriptions is a more complicated task since a single triple may partially resolve a query only by satisfying one of its sub-queries. Moreover, as the triples satisfying the query are not necessarily inserted in the network at the same time, nodes need to keep traces of queries that have been already partially satisfied and create notifications only when all sub-queries are satisfied.

Dynamic Semantic Space. In [42], Gu *et al.* propose Dynamic Semantic Space (DSS), a schema-based P2P overlay network where RDF statements are stored based on their semantics. Peers are organized in a ring structure enabling the mapping from a $\mathtt{k}$-dimensional semantic space into a one dimensional semantic space. Peers are grouped into clusters, and clusters having the same semantics are organized into the *same semantic cluster*. While there is no constraint on the overlay topology within the semantic clusters, they themselves form a Chord [83] overlay structure. Each cluster is identified by a `clusterID` given by a $\mathtt{k}$-bit binary string such as $\mathtt{k} = \mathtt{x} + \mathtt{y}$. While the $\mathtt{x}$ first bits identify the semantic cluster, denoted $\mathtt{SC}$, the $\mathtt{y}$ bits represent the identifier of a cluster $\mathtt{c}$ belonging to $\mathtt{SC}$. Therefore, the semantic space infrastructure can have a maximum of $2^{\mathtt{x}}$ semantic clusters and $2^{\mathtt{y}}$ clusters per $\mathtt{SC}$. As an example, Figure 3 depicts the architecture of a dynamic semantic space where 4 semantic clusters, $\mathtt{SC}_1, \mathtt{SC}_2, \mathtt{SC}_3, \mathtt{SC}_4$, with 16 clusters in total form the semantic space infrastructure. Peers maintain a set of neighbors in their adjacent clusters as well as in other semantic clusters in order to enable inter-clusters and intra-clusters communication. For instance, as $\mathtt{C}_2$ and $\mathtt{C}_{16}$ are the two adjacent clusters of cluster $\mathtt{C}_1$ where the peer $\mathtt{p}_0$ belongs to, $\mathtt{p}_0$ maintains references to peers $\mathtt{p}_2$, $\mathtt{p}_3$ such as $\mathtt{p}_3 \in \mathtt{C}_2$ ($\mathtt{C}_2 \in \mathtt{SC}_1$) and $\mathtt{p}_2 \in \mathtt{C}_{16}$ ($\mathtt{C}_{16} \in \mathtt{SC}_4$). Moreover, $\mathtt{p}_0$ maintains pointers to other semantic clusters, that is $\mathtt{SC}_2$ and $\mathtt{SC}_3$. DSS provides a load balancing mechanism by controlling the number of peers per cluster and thus the global

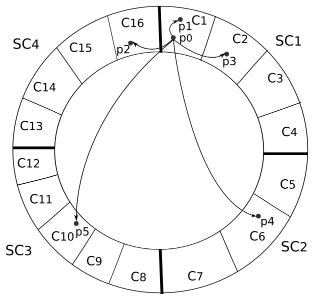


Fig. 3. Example of a Dynamic Semantic Space architecture with 4 semantic clusters and 16 clusters

load. Therefore, a splitting and merging process can occur if the number of nodes per cluster reaches a threshold value.

Data indexing. Authors propose a two-tier ontology-based semantic clustering model: the upper layer defines a set of concepts shared by all peers. A peer needs to define a set of low-layer ontologies and store them locally. Therefore, RDF statements which are semantically similar are grouped together and can thus be retrieved by queries having the same semantics. To join the network, a peer has to merge the upper layer ontology with its local ontologies and create RDF data instances which will be added to the merged ontology to form its local knowledge base. A mapping of its RDF data to one or more semantic clusters is performed and the peer joins the most semantically related cluster.

Query processing. The query processing mechanism is performed in two steps. First, when the query, expressed in RDF-Data Query Language (RDQL), is received, the peer pre-processes the query to get the information about the semantic cluster `SC`. Second, when the query reaches the targeted `SC`, it will be flooded to all peers within `SC`.

Subscription processing. DSS also supports a content-based pub/sub mechanism. Once a subscription request is generated it will be mapped to the corresponding `SC`. From there, it will be forwarded to every node within `SC`. Upon receipt of the subscription request, a peer checks for a local matching of the subscription against its local RDF data. After a successful matching, it will send back the results to the subscriber(s). If a modification occurs on the local data, the producer peer notifies subscriber node(s) by sending notifications that follow exactly the reverse path of the corresponding subscription. Whenever a peer wants to unsubscribe to a query, an unsubscription request will be sent directly to the relevant producers.

PAGE. Put And Get Everywhere (PAGE) presented in [33] is a RDF distributed repository based on Bamboo DHT [76]. The RDF data model introduced in this work extends the standard RDF data model by introducing the notion of `context`. This concept is application-dependent. For instance, in the information integration use case, the context is the URI of the file or the repository from which a RDF triple originated. In other scenarios, RDF triples sharing the same semantics can have the same context. Therefore, in PAGE, each RDF triple, denoted by `t=(s,p,o)`, is associated with a context, labeled `c`. The RDF triple combined with the context forms what is called a `quad`. Each quad is indexed six times and identified by an ID that is built by concatenating the hash values of its elements `(s,p,o,c)`.

Data indexing. PAGE associates to each index an `index code`. This field identifies the used index (i.e., {`spoc,cp,ocs,poc,csp,os`}) for building the quad ID. For instance, suppose that one wants to store the quad `quad=(x,y,z,w)`. Assume that the hashed values of its elements are given by `hash(x)=1`, `hash(y)=F`, `hash(z)=A`, `hash(w)=7`; and indexes codes by: `idx(spoc)=1`, `idx(cp)=4`, `idx(ocs)=6`, `idx (poc)= 9`, `idx(csp)= C`, `idx(os)=E`. The concatenation of `'1FA7'` with `idx (spoc)=1` results to `ID='11FA7'`. The quad will then be stored on the node that has the numerically closest identifier to `ID`. The same operation is performed for each of these six indexes giving at each time the node identifier where the quad will be stored.

Query processing. The query processing algorithm starts by associating to each access pattern specified in the query an identifier ID as explained earlier. A *mask* property is used to specify the number of elements fixed in the query. Moreover, the query is controlled by the number of *hops* which denotes the number of digits from the query ID that have been already considered for the query evaluation during the routing process. Take back the example discussed above and consider the access pattern `q=(s,p,?,?)` which looks, for the given subject `s` and predicate `p`, for all object values for any context. This access pattern requires the index (`idx(spoc)=1`) to be resolved. For unknown parts a sequence of `0` is assigned. The query will then be converted to `11F00/3`. The mask value here is `3` meaning that the first three digits of the identifier are fixed. Thereby, only the *object* and the *context* will be considered during the query processing.

3.2 Cube-Based Approaches

Edutella. Nejdl *et. al* propose a RDF-based P2P architecture called Edutella [72] which makes use of super-peers. In this kind of topology, a set of nodes are selected to form the super-peer network, building up the "backbone" of the P2P network, while the other peers connect in a star-like topology to those super-peers. Super-peers form a so-called *HyperCuP* topology [80] and are responsible for the query processing. In this organization, each super-peer can be seen as the root of a spanning tree which is used for query routing, broadcasting and indices updating. Each peer sends indices of its data to its super-peer. Edutella implements a schema-based P2P system where the system is aware of schema information and takes it into consideration for query optimization and routing. Data indices can be stored in different granularities, like schema, property, property value or property value range. However, this index contains peer indices instead of data entries. This kind of index is called a `Super-Peer-Peer`

(`SP-P`) index. For example, a query with a given schema received by a super-peer will be only forwarded to peers supporting that schema. This allows to narrow down the number of peers to which the query will be broadcasted. In addition to the `SP-P` index, the super-peers maintain `Super-Peer-Super-Peer (SP-SP)` indices in order to share their index information with other super-peers. Accordingly, queries are forwarded to super-peer neighbors based on those indices. Edutella supports the RDF Query Exchange Language (RDF-QEL).

Data indexing. When a new peer registers to a super-peer, labeled `sp`, it advertises the necessary schema information to `sp`. `Sp` checks this new schema against its `SP-P` index entries. If there is new information that has to be added in the `SP-P` index, `sp` broadcasts the new peer advertisement to its super-peers neighbors in order to update their `SP-SP` indices. In a similar way, indexes have to be updated whenever a peer leaves the network.

Query processing. Query routing in Edutella is improved by using *similarity-based clustering strategy* at the super-peer level to avoid as much as possible the broadcasting during the query routing phase. This approach tries to integrate new peers with existing peers that have similar characteristics based, for example, on a topic ontology shared by them.

RDFCube. Monato *et al.* propose RDFCube [67], an indexing scheme of RDF-Peers (Section 3.1) based on a three-dimensional CAN coordinate space. This space is made of a set of cubes, with the same size, called *cells*. Each cell contains an *existence-flag*, labeled `e-flag`, indicating the presence (`e-flag=1`) or the absence (`e-flag=0`) of a triple within the cell. The set of consecutive cells belonging to a line parallel to a given axis forms a *cell sequence*. Cells belonging to the same plane perpendicular to an axis form the *cell matrix*.

Data indexing. Once a RDF triple `t=(s,p,o)` is received, it will be mapped to the cell where the point `p` with the coordinates `(hash(s),hash(p),hash(o))` belongs to.

Query processing. As for RDF triples, the query is also mapped into a cell or a plane of RDFCube based on the hash values of its constant part(s). Consequently, the set of cells including the line or the plane where the query is mapped are the candidate cells containing the desired answers. Note that RDFCube does not store any RDF triples but it stores bit information in the form of e-flags. The interaction between RDFCube and RDFPeers is as follows: RDFCube is used to store `(cell matrixID, bit matrix)` pairs such as the `matrixID` is a matrix identifier and represents the key in the DHT terminology, while `bit matrix` is its associated value. RDFPeers stores the triples associated with the bit information. This bit information is basically used to speed up the processing of a *join query* by performing an `AND` operation between bits and transferring only the relevant triples. As a result, this scheme reduces the amount of data that has to be transferred among nodes.

3.3 Tree-Based Approaches

GridVine. GridVine [31] is a distributed data management infrastructure based on P-Grid DHT [8] at the overlay layer and which maintains a *Semantic Mediation Layer* on top of it as it is shown in Figure 4. GridVine enables distributed search, persistent storage and semantic integration of RDF data.The upper layer takes advantage of the overlay architecture to efficiently manage heterogeneous data including schemas and schema mappings.

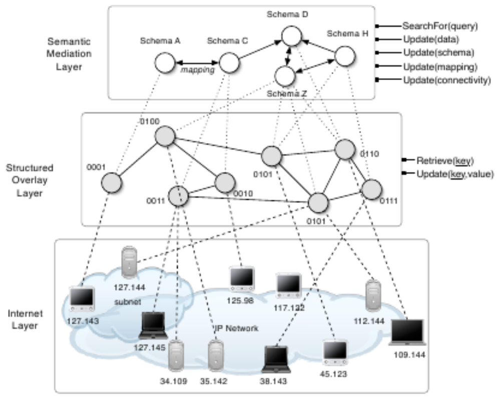


Fig. 4. GridVine architecture incarnated in the semantic mediation layer and the structured overlay layer (taken from [31])

Data indexing. As in RDFPeers [23], GridVine indexes triples three times at the P-Grid layer based on their subjects, objects and predicates. As an example, for a given triple `t`, the `insert(t)` operation results into `insert(hash(s), t)`, `insert(hash(p), t)` and `insert(hash(o), t)`. In that way, a triple inserted at the mediation layer triggers three insert operations at the overlay layer. GridVine also supports the sharing of schemas. Each schema is associated with a unique key which is the concatenation of the logical address of the peer posting the schema, say π(`p`), with the schema name, `Schema_Name`. Therefore, a schema indexing operation may take the form of `Update(`π`(p) : hash(Schema_Name), Schema _Defintion)`. In order to integrate all semantically related yet syntactically heterogeneous data shared by peers at the P-Grid level, GridVine supports the definition of pairwise semantic mappings. The mapping allows the reformulation of a query against a given schema into a new query posed against a semantically similar schema. The schema mapping uses Web Ontology Language (OWL) [12] statements to relate two similar schemas.

Query processing. Lookup operations are performed by hashing the constant part(s) of the triple pattern. Once the key space is reached, the query will be forwarded to peers responsible for that key space. As in P-Grid, the search complexity of atomic triples pattern is $\mathcal{O}(\texttt{log}(\Pi))$ such as Π is the entire key partition.

Note that atomic queries where none constant part is fixed are not supported by the query processing scheme. GridVine is also able to handle disjunctive and conjunctive queries by iteratively resolving each triple pattern in the query and perform distributed joins across the network. Therefore, the system query processing capabilities are very similar to RDFPeers, but GridVine takes also into consideration the schema heterogeneity which is not addressed by RDFPeers.

UniStore. Karnstedt *et al.* present their vision of a universal data storage at the Internet-scale for triples' repository through the UniStore project [49]. Their solution is based on P-Grid [8], on top of which they built a triple storage layer that enables the storage of data as well as metadata.

Data indexing. Authors chose to adopt a universal relation model allowing schema-independent query formulation. In order to speed up and take advantage of the underlying features of the DHT for fast lookups, all attributes are indexed. When dealing with relational data, each tuple $(\mathtt{OID}, \mathtt{v}_1, \ldots, \mathtt{v}_x)$ of a given relation schema $\mathtt{R}(\mathtt{A}_1, \ldots, \mathtt{A}_x)$ is stored in the form of $\mathtt{x}$ triples: $(\mathtt{OID}, \mathtt{A}_1, \mathtt{v}_1), \ldots, (\mathtt{OID}, \mathtt{A}_x, \mathtt{v}_x)$ where `OID` is a unique key, $\mathtt{A}_i$ an attribute name and $\mathtt{v}_i$ its corresponding value. The `OID` field is used to group a set of triples for a logical tuple. Each triple is indexed on the `OID`, the concatenation of $\mathtt{A}_i$ and $\mathtt{v}_i$ ($\mathtt{A}_i\#\mathtt{v}_i$) and finally on the value $\mathtt{v}_i$.

Query processing. UniStore allows the use of a query language called Vertical Query Language (VQL) which directly derives from SPARQL. This language supports DB-like queries as well as IR-like (Information Retrieval) queries. It comes with an algebra supporting traditional "relational" operators as well as operators needed to query the distributed triple storage. It offers basic constructs such as `SELECT`, `ORDER BY`, etc. as well as advanced ones such as `SKYLINE OF`. Furthermore, in order to support large-scale and heterogeneous data collection, the language was extended with similarity operators (e.g., similarity join) and ranking operators (e.g., skyline queries). Operators can be applied to all levels of data (e.g., instance, schema and metadata). Each of these logical operators have a "physical" implementation which only rely on functionality provided by the overlay system (key lookup, multicasts, etc.) and they differ in the kind of used indexes, applied routing strategy, parallelism, etc. These physical operators are used to build complex query plans, which in turn are used to determine worst-case guarantees (almost all of them being logarithmic) as well as predict exact costs. This allows the system to derive a cost model for choosing concrete query plans, which is repeatedly applied at each peer involved in the query resolution and thus resulting in an adaptive query processing approach.

3.4 Generic DHT-Based Approaches

Battré et al. In [16], authors propose a data storage and retrieval strategy for DHT-based RDF store. They use similar indexing mechanisms as in [9, 23, 67], the proposed approach indexes the RDF triple by hashing its subject, predicate and object. It takes into consideration RDFS reasoning on top of DHTs by applying RDFS reasoning rules.

Data indexing. The main characteristic of this approach compared to others is that nodes host different RDF repositories, making a distinction between local and incoming knowledge. Each RDF repository is used for a given purpose:

- **Local triples repository** stores triples that originate from each node. A local triple is disseminated in the network by calculating the hash values of its subject, predicate and object and sending it to nodes responsible for the appropriate parts of the DHT key space.
- **Received triples repository** stores the incoming triples sent by nodes.
- **Replica repository** ensures *triple availability* under high peer churn. The node with the numerically closest ID to the hash value of a triple becomes the root node of the replica set. This node is responsible for sending all triples in its received repository to the replica nodes.
- **Generated triples repository** stores triples that are originated from applying forward chaining rules on the received triples repository, and they are then disseminated as local triples to the target nodes. This repository is used for RDFS reasoning.

In order to keep the content of the received triples repository up-to-date, specially under node leaving or crashing, triples are associated with an expiration date. Therefore, the peer responsible of that triple is in charge of continuously sending update messages. If the triple expiration time is not refreshed by the triple owner, it will be eventually removed from these repositories. This approach takes care of load balancing issues in particular for uneven key distribution. For instance, the DHT may store many triples with the same predicate `‘rdf:type’`. As the subject, predicate and object will be hashed, the node responsible for the `hash(rdf:type)` is a target of a high load. Such situation is managed by building an overlay tree over the DHT in order to balance the overloaded nodes.

Query processing. In another work [45], one of the authors proposes a RDF query algorithm with optimizations based on a look-ahead technique and on Bloom filters [21]. Knowledge and queries are respectively represented as *model* and *query* directed graphs. The query processing algorithm basically performs a matching between the query graph and the model graph. On one side, there is the candidate set which contains all existing triples, and on the other side, there is a candidate set containing the variables. These two sets are mutually dependent, therefore a refinement procedure has to be performed to retrieve results for a query. This refinement proceeds in two steps. First, starting from the variable's candidate set, a comparison is done with the candidate sets for each triple where the variable occurs. If a candidate does not occur within the triple candidate set, it has to be removed from the variable candidate set. The second step goes the other way around, that is, it looks at the candidate set for all the triples and removes every candidate where there is no matching value within the variable's candidate set.

The look-ahead optimization aims at finding better paths through the query graph by taking into account result set sizes per lookup instead of the number of lookups. This yields fewer candidates to transfer but the trade-off is that it incurs more lookups.The second optimization, using Bloom filters, works as follows: consider candidates for a triple $(\mathtt{a}, \mathtt{v}_2, \mathtt{v}_3)$, where `a` is a fixed value and $\mathtt{v}_2$ and $\mathtt{v}_3$ are variables. When retrieving the candidates by looking up using the fixed value `a`, it may happen that the querying node might already have candidates for the two variables. The queried node can reduce the results' sets with the knowledge of sets $\mathtt{v}_2$ and $\mathtt{v}_3$. However, those sets may be large, for this reason the authors use Bloom filters to reduce the set representation. The counterpart of using Bloom filters, is that they yield false positives, i.e., the final results sets which will be transferred may contain non-matching results. A final refinement iteration will be done locally which will remove those candidates and thus ensuring the correctness of the query results.

Query Chain and Spread by Value Algorithms. In [59], Liarou *et al.* propose the Query Chain (QC) and Spread by Value (SBV) algorithms to evaluate conjunctive queries over structured overlays.

QC - Data indexing. As in RDFPeers [23], in QC algorithm, a RDF triple is indexed to three nodes. More precisely, for a node that wants to publish a triple `t` such as `t=(s,p,o)`, the index identifiers of `t` is computed by applying a hash function on `s`, `p` and `o`. Identifiers `hash(s)`, `hash(p)` and `hash(o)` are used to locate the nodes that will then store the triple `t`.

QC - Query processing. In this algorithm, the query is evaluated by a *chain* of nodes. Intermediate results flow through the nodes of this chain and the last node in the chain delivers the result back to the query's originator. More precisely, the query initiator, denoted by `n`, issues a query `q` composed of $\mathtt{q}_1$, $\mathtt{q}_2, \ldots, \mathtt{q}_\mathtt{i}$ patterns and forms a *query chain* by sending each triple pattern to a (possibly) different node, based on the hash value of the constant part(s) of each pattern. For each of the identified nodes, the message $\mathtt{QEval}(\mathtt{q}, \mathtt{i}, \mathtt{R}, \mathtt{IP}(\mathtt{x}))$ will be sent such that `q` is the query to be evaluated, `i` the index of the pattern that will managed by the target node, `IP` the address of the query's originator `x` and `R` an intermediate relation to hold intermediate results. When there is more than one constant part in the triple pattern, subject will be chosen over object over predicate in order to determine the node responsible for resolving this triple. While the query evaluation order can greatly affect the algorithm performance including the network traffic and the query processing load, the authors adopt the default order for which the triple patterns appear in the query.

SBV - Data indexing. In the SBV algorithm, each tripl $\mathtt{t} = (\mathtt{s}, \mathtt{p}, \mathtt{o})$ is stored at the successor nodes of the identifiers `hash(s)`, `hash(p)`, `hash(o)`, `hash(s+p)`, `hash(s+o)`, `hash(p+o)` and `hash(s+p+o)` where the `'+'` operator denotes the concatenation of string values. Using triple replication, the algorithm aims to achieve a better query load distribution at the expense of more storage space.

SBV - Query processing. SBV extends the QC algorithm in the sense that the query is processed by multiple chains of nodes. Nodes at the leaf level of these chains will send back results to the originator. More precisely, a node posing a conjunctive query q in the form of $\mathtt{q}_1 \wedge \ldots \wedge \mathtt{q}_\mathtt{k}$ sends `q` to a node $\mathtt{n}_1$ that is able to evaluate the first triple pattern $\mathtt{q}_1$. From this point on, the query plan produced by SBV is created dynamically by exploiting the values of the matching triples that nodes find at each step. As an example, a node $\mathtt{n_1}$ will use the values found locally that matches $\mathtt{q_1}$, to bind the variables of $\mathtt{q_2} \wedge \ldots \wedge \mathtt{q_k}$ that are in common with $\mathtt{q_1}$ and produce a new set of queries that will jointly determine the answer to the query's originator. Unlike the query chain algorithm, to achieve a better distribution of the query processing load, if there are multiple constants in the triple pattern, the concatenation of all constant parts is used to identify nodes that will process the query.

The performance of both QC and SBV algorithms can be improved through the caching of the IP addresses of contributing peers. This information can be used to route similar queries and thus reduce the network traffic and query response time. A similar algorithm is presented in [14] where conjunctive queries are resolved iteratively: starting by a single triple pattern and performing a lookup operation to find possible results of the current active triple pattern. These results will be extended with the resolution of the second triple pattern and an intersection of the current results with previous ones will be done. The procedure is repeated until all triple patterns are resolved. Unlike Query Chain algorithm discussed in [59], where intermediate results for a conjunctive query resolution are usually sent through the network to nodes that store the appropriate data, Battré in [14], combines both the fetching and the intermediate results forwarding approaches. Such decision is taken at runtime and is based on the estimated traffic for each data integration techniques.

3.5 P2P-Based Publish/Subscribe Systems for RDF Data Storage and Retrieval

Cai et al. Cai *et al.* have proposed in [24] a content-based pub/sub layer extension atop their RDFPeers infrastructure [23] (Section 3.1). In their system, each peer $\mathtt{n}$ maintains a list of local subscriptions, along with the following information: triple patterns, the subscriber node identifier, a requested notification frequency and the requested subscription expiration date. Once the node $\mathtt{n}$ stores (upon insertion) or removes a triple locally, it evaluates the matching subscription queries. This is also done after a triple update or when a collection of several matches for a given subscription is made. Afterwards, $\mathtt{p}$ notifies the subscriber by sending matched triples.

Subscription processing. The subscription mechanisms support *atomic, disjunctive, range* and *conjunctive multi-predicate* queries[1]. The basic subscription mechanism for *disjunctive* and *range* queries is similar to the one used for atomic queries, except that a subscription request is stored by all nodes that fall into the hashed identifiers of the lower and the upper range bounds. The subscription scheme to manage *conjunctive multi-predicate* subscriptions is similar to the query chain algorithm discussed in Section 3.4. Initially, the subscription request $\mathtt{S_r}$ will be routed to the node $\mathtt{n}$ corresponding to the first triple pattern $\mathtt{s}$ in the

[1] Not implemented at the time of the writing of their paper and some combinations of conjunctive and disjunctive queries are not supported.

subscription. Once the node `n` processes the first conjunct `s`, it removes `s` from $\mathtt{S_r}$ and stores it locally. In addition, the node `n` also stores the hash value of the next pattern in $\mathtt{S_r}$. The node, receiving the subscription request (i.e., $\mathtt{S_r} \setminus \{\mathtt{s}\}$) sent by `p` stores the next pattern and routes the remaining patterns towards the appropriate peer and so on and so forth. The node that stores the last pattern in the subscription $\mathtt{S_r}$ will also store the subscription request's originator. Therefore, whenever new triples are matched by the first pattern, they will be forwarded to the second node in the chain. The second and subsequent nodes will only further forward those triples if they also match their local filtering criterion. Moreover, the authors propose an extension to support highly skewed subscriptions patterns, to avoid having a majority of nodes with few if any subscribers while a handful of node attracts a lot of subscribers. This extension promotes a proportional notification's dissemination scheme to the number of subscribers. The authors provide two ways to do so: (`i`) similar subscriptions from different subscribers are aggregated and reduced into a single entry with multiple subscribers and (`ii`) when a certain threshold of subscribers is exceeded, a node will maintain a "repeater nodes" list which is used to distribute the load of the notification propagation among nodes. In order to cope with churn and failure, a replication mechanism, directly inherited from MAAN [22], along with a repair protocol for the subscriptions and the data is provided but only for atomic queries. Replication can be tuned using a `Replica_Factor` parameter, thus, the subscription list will be replicated to the next `Replica_Factor` nodes in the identifier space. The same principle can be applied to RDF triples. To manage situations where a user leaves while his subscriptions are still active, each subscription is characterized by a *maximum duration parameter* which controls its lifetime.

Chirita et al. In [10], Chirita *et al.* propose a set of algorithms to manage publications, subscriptions and notifications on top of super-peer like topology as in Edutella [72] (Section 3.2). The authors formalized the basic concepts of their content-based pub/sub system using a *typed first-order* language. Such formalization enables a precise reasoning on how subscriptions, advertisements and publications are managed in such systems. The formalism is further used by the authors to introduce the *subscription subsumption* optimization technique, which we will explain hereafter.

Advertisement indexing and processing. In this model, once a peer `n` connects to its super-peer node, denoted `sp`, it compulsorily advertises the resources it can offer in the future by sending advertisements to `sp`. These advertisements are useful to super-peers in order to build *advertisement routing indices* that will further be used for subscriptions processing. Advertisements contain one or more elements which can either be a schema identifier (e.g.,`<dc>`,`<lom>`), a property/value pair (e.g.,{`<dc:year>,2010`}) or a single property (e.g.,`<dc:year>`). Consequently, three levels of indexing are managed:

– **Schema level.** This level of indexing contains information about RDF schemas supported by the peers. Each schema is uniquely identified by a

URI. The schema indexing can be used, for example, to constrict the subscriptions forwarding only to the peers supporting that schema.

– **Property/Value level.** This second level indexes peers providing the resources rather than the property itself. It is mainly used to reduce the network traffic.
– **Property level.** This level of indexing is useful when a peer chooses to use specific properties/attributes from one or a set of schemas. Thus, the property index contains properties, identified by `namespace/schemaID` in addition to the property name. Each property points to one or more peers that support them.

Advertisements are selectively broadcasted from a super-peer to its super-peer neighbors. The advertisement update process is triggered if a peer joins or leaves the network or whenever the data that it is responsible for is modified. Therefore, references associated to peers and indexed at the super-peer level also have to be updated.

Subscription processing. Each super-peer manages a subscription in a partially ordered set (poset). Subscriptions sent by a peer `n` are inserted into the poset of its super-peer `sp`. As in the SIENA system [25], the poset consists of a set of subscriptions. The subscriptions' poset is actually a hierarchical structure of subscriptions which captures the concept of subscription subsumption, that is, when the result of a subscription $\mathtt{q}_1$ is a subset of the result subscription $\mathtt{q}_2$, we say that $\mathtt{q}_2$ *subsumes* $\mathtt{q}_1$. When new subscriptions are subsumed by previously forwarded ones, the super-peer does not forward these new subscriptions to its super-peer neighbors. However, consider an advertisement $\mathtt{Adv}_1$ sent by $\mathtt{sp}_2$ to $\mathtt{sp}_1$. If $\mathtt{sp}_1$ has already received a subscription `S` (by another peer) and there is a match with $\mathtt{Adv}_1$, then $\mathtt{sp}_1$ informs $\mathtt{sp}_2$ not to forward any advertisement of this kind anymore, because the subscription responsible for `S` subsumes $\mathtt{Adv}_1$.

When a new notification *not* arrives at a super-peer `sp`, `sp` checks for the subscriptions satisfying *not* in its local subscriptions poset in order to forward them to the subscriber nodes. This approach handles the notifications under peer churn. When a peer `n` associated to a super-peer `sp` leaves the network, `sp` puts the subscriptions of `n` in a buffer for a period of time in order to receive notifications sent to `n`. Once `n` reconnects to `sp`, these notifications will then be forwarded to it. The authors also make use of cryptographic algorithms to provide unique identifiers to peers. After a leave (or a crash), a node may have a different IP address. This unique identifier will be used to retrieve waiting subscription results.

Single and Multiple Query Chain Algorithms. Liarou *et al.* have proposed in [58] a content-based pub/sub system, which mainly focuses on conjunctive multi-predicate queries. They provide two *DHT-agnostic* algorithms, namely, the *Single Query Chain* (SQC) and the *Multiple Query Chains* (MQC), that extends the Query Chain algorithm detailed in Section 3.4.

Subscription processing. In their pub/sub system, each triple $\mathtt{t}$ is characterized by a *published time* parameter, labeled $\mathtt{pubT(t)}$. Moreover, each subscription $\mathtt{S}$ such as $\mathtt{S} = \mathtt{s_1} \wedge \mathtt{s_2} \wedge ... \wedge \mathtt{s_n}$ is identified by a $\mathtt{key(S)}$ and has a timestamp indicating the *subscription time*, denoted by $\mathtt{subscrT(S)}$. As a result, $\mathtt{s_i}$ is also characterized by a subscription time such as $\mathtt{subscrT(s_i)} = \mathtt{subscrT(S)}$. Therefore, a triple $\mathtt{t}$ can satisfy a subscription $\mathtt{s}$ of $\mathtt{S}$ *iff* $\mathtt{subscrT(s)} < \mathtt{pubT(t)}$. Subscriptions and notifications forwarding policies are similar to algorithms proposed in [59]. In the case of SQC, for each query, a single query chain is created at node $\mathtt{r}$ upon receipt of query $\mathtt{S}$ whereas MQC goes a step further. In MQC, first, a query $\mathtt{S}$ is indexed to a single node $\mathtt{r}$ according to one of $\mathtt{S}$'s sub-queries. Then, each time a triple arriving at $\mathtt{r}$ satisfies this sub-query, the subject is used to rewrite $\mathtt{S}$ and thus, for each different rewritten query, a query chain is created, yielding multiple query chains. This has the benefit of achieving a better load distribution than SQC. For optimization purposes, the authors propose a query clustering mechanism where similar subscriptions are grouped together. For instance, triples which have been indexed on node $\mathtt{n}$ using the same predicate $\mathtt{p}$ will be answered when a new triple with the predicate $\mathtt{p}$ is inserted. In such a scenario, only one matching operation has to be performed whenever such a triple is inserted.

Continuous Query Chain and Spread-By-Value Algorithms. In another work [57], Liarou *et al.* have proposed an extension of their two algorithms presented in [59] (QC and SBV). By introducing a *continuous* version of these algorithms, thus becoming CQC (*Continuous Query Chain*) and CSBV (*Continuous Spread By Value*).

CQC - Subscription processing. A node $\mathtt{n}$ that wants to subscribe to a conjunctive query $\mathtt{q} = [\mathtt{q_1}, \ldots, \mathtt{q_k}]$ will do so by indexing each triple pattern $\mathtt{q_j}$ to a different node $\mathtt{n_j}$. Thus, each of these nodes will be responsible for processing their part of the query $\mathtt{q_j}$ and will be a part of a *query chain* of $\mathtt{q}$. Each node indexes the triple patterns as in QC, explained in Section 3.4. When a node receives a newly indexed triple $\mathtt{t}$, it will check any relevant indexed queries in its *query table (QT).* If any match is found, there is a *valuation*[2] $\mathtt{v}$ over the variables of $\mathtt{q_j}$ such that $\mathtt{v(q_j)} = \mathtt{t}$. (i). If it is the first node in the chain, it forwards the valuation $\mathtt{v}$ (holding a partial answer to $\mathtt{q}$) to the next node in the chain, $\mathtt{n_i}$. When receiving this *intermediate result*, $\mathtt{n_i}$ will apply the received valuation and compute a new valuation $\mathtt{w}$ to the pattern $\mathtt{q_i}$ it holds, resulting in $\mathtt{q_i'} = \mathtt{w(q_i)}$. Then, it will try to find triples matching $\mathtt{q_i'}$ in its *triple table (TT)* that have already arrived. If there is a match, a new intermediate result is produced, a new valuation $\mathtt{w'} = \mathtt{w} \cup \mathtt{v}$. This new valuation is then passed along the query chain and stored in an *intermediate result table (IRT)*, which is used when new triples arrive. When the last node in the chain receives a set of intermediate results, it will check its *TT* and if matches are found, an *answer* to the query $\mathtt{q}$ is sent back to the node that originally posed $\mathtt{q}$. (ii). If the node is not the first in the query chain, it will store the new triple in its *TT* and search in its *IRT* to see if

[2] Concept used to denote values satisfying a query.

an evaluation of the query that has been suspended can now continue due to `t` that has just arrived. For each intermediate result `w` found, a new valuation $\mathtt{w}'$ is computed and forwarded to the next node in the query chain.

CSBV -Subscription processing. This algorithm actually extends CQC in a way that it achieves a better distribution of the query processing load. In CQC, a query chain with a fixed number of participants is created upon submission of a query `q`, whereas in CSBV no query chain is created. Instead `q` is indexed *only to one node* which will be responsible for one of the triple patterns of `q`. A node `n` can use the valuation `v` to rewrite $\mathtt{q} = [\mathtt{q}_1, \ldots, \mathtt{q}_k]$ with fewer conjuncts $\mathtt{q}' = [\mathtt{v}(\mathtt{q}_2), \ldots, \mathtt{v}(\mathtt{q}_n)]$ and decides *on the fly* the next node that will undertake the query processing. Because $\mathtt{q}'$ is conjunctive like `q`, its processing proceeds in a similar manner. Depending on the triples that trigger $\mathtt{q_i}$, a node can have multiple next nodes for the query processing. Thus, the responsibility of evaluating the next triple pattern of `q` is distributed to multiple nodes compared to just one node in CQC, leading to a better load distribution. Query indexing follows the same heuristics used in CQC, with the difference that if the query has multiple constants, a combination of all the constant parts will be used to index the query, following the triple indexing of SBV presented in Section 3.4.

When receiving a new triple `t`, this basically proceeds as in CQC (in terms of checking if a match is found for the pattern) with the only exception that the forwarding of the intermediate results is dynamic because computed on the fly. Intermediate results are processed in the same logic as in CQC with the difference that instead of forwarding a single intermediate result to the next node, a *set* of intermediate results is created and delivered possibly to different nodes. These two algorithms come with communication primitives that enable messages to be sent in bulk. Naturally, each step of these algorithms comes at a certain cost, but some improvement techniques are reused from QC and SBV such as IP caching used to maintain the address of the next node in the chain the peer should send the intermediate results to, or also a query chain optimization scheme which tries to find an optimized nodes' order based on specific metrics such as the rate of published triples that trigger a triple pattern.

Ranger et al. In [74], the authors introduce an information sharing platform based on Scribe [26], a topic-based pub/sub system with which they use queries as topics. Interestingly, the algorithm they propose does not index data *a priori*. Instead, their scheme relies upon finding results from scratch with redundant caching and cached lookups mechanisms. This is done by taking advantage of the Scribe infrastructure, that is, the possibility to subscribe to a topic and to lookup within the group of the topic. Frequent queries are cached effectively as they occur, remaining active as long as clients are reading from it. Queries, which can be either atomic or complex, are expressed in a SPARQL dialect.

Subscription processing. Queries are translated into trees of either atomic or complex sub-queries. An atomic query is a simple, independent query that all peers can execute on their local content since it does not depend on the results from other peers. A complex query resorts to results from other queries to

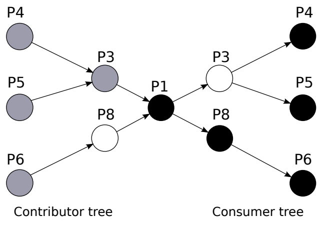


Fig. 5. Contributor and consumer pipelines resulting from `P1` performing a query. Gray nodes are contributors to query `Q`; white nodes are forwarders, that is, nodes which help in the dissemination even if they are not interested in the content; black nodes are consumers of the results integrated by `P1` (figure taken and redrawn from [74]).

produce its results, either by aggregation, filtering or calculation. Since all peers can execute arbitrary complex queries, the exact distinction between atomic and complex queries is somehow fuzzy. It is usually unfeasible to determine if the query should be run locally. This depends on whether objects are stored by a single peer, or if parts of objects are scattered across different peers. This also depends on the application built on top of this scheme. In order to circumvent this ambiguity, the algorithm co-locates predicates from designated namespaces (e.g., `dc,rdf,type,etc.`). This means that for a same subject, all predicates from a co-located namespace will be available on the same peer. When a peer performs a query, it first looks up in the group for a peer `p` that already knows (or is interested to know) the query's result. If `p` is found, the peer joins the so called *consumer* pipeline tree rooted at a producer of the query results. When a peer matches an atomic query, it establishes a *contributor* pipeline tree rooted at itself. When consumers ask for results, the producing peer will read/combine results from the contributor pipeline and forward them to the consumers. Figure 5 shows the result pipelines established when a peer `P1` performs a query. Within a group, whenever a peer is processing an atomic query, it will broadcast the fact to other peers.When a peer receives a broadcast message, if it has relevant content or if the query is *live*, that is if the originator wants to be able to wait for new results as they happen, it will be forced to contribute. The sole reason for this is to avoid ignoring any potential source of results. The algorithm makes use of Pastry's built-in mechanisms [78] to become aware of intentional or accidental departures, and the contributing or forwarding peer will be asked to re-send the message and thus repairing the tree with minimal losses.

4 Summary and Discussions

The combination of P2P technologies with RDF data has become a very active research area aiming at sharing and processing large amounts of data at the scale of the Internet. In addition to the inherent challenges related to such large scale infrastructure (e.g., network partition, topology maintenance, fault-tolerance, etc.), enabling complex querying of RDF data on top of such infrastructures requires advanced mechanisms, especially from a data indexing and query processing point of views. The presented works are summarized in Table 2 and hereafter we will discuss some challenges surrounding this combination and point out some improvements with respect to the underlying overlays, the way the RDF data is indexed and finally how queries are processed.

Table 1. Comparison of the different presented approaches in terms of indexing mechanism, the routing scheme and the data replication property

Topology		**Hashing**	**Routing scheme**	**Replication**
Ring	RDFPeers [23]	✓	Index-based	✓
	Atlas [54]	✓	Index-based	✓
	DSS [42]	✗	Semantic mapping	✗
	PAGE [33]	✓	Index-based	✓
Cube	Edutella [71]	✗	Semantic mapping	✗
	RDFCube [67]	✓	Index-based	
Tree	GridVine [9]	✓	Semantic mapping and Index-based	✓
	Unistore [49]	✓	Index-based	✓
Generic DHT	*Battré et al.* [16, 45]	✓	Index-based	✓
	QC & SBV [59]	✓	Index-based	✓

Overlay topology. As we have already seen, a lot of work has been carried out towards a scalable distributed infrastructure of RDF data storage and retrieval. Most of the presented approaches choose the structured P2P networks as infrastructure basis (e.g., [9, 16, 23, 33, 42, 44, 54, 59, 67, 72]). Such choice was motivated by the fact that, first, as it is already argued in [62], structured P2P overlays provide a practical substrate for building scalable distributed applications due to the guarantees that they provide. Secondly, the structural nature of the RDF data model influences the choice of the topology as it can be noticed through the data indexing model of most of the presented approaches. We detailed how each work indexes RDF data on top of their overlay and associated topology (that is why we chose a topology oriented classification in Section 3). However there is no clear evidence from the state of the art study that a specific overlay topology is the most suitable for handling RDF data.

Data indexing. As we have already seen, a lot of work has been carried out towards a scalable distributed infrastructure of RDF data storage and retrieval. Most of the presented approaches choose the structured P2P networks as infrastructure basis (e.g., [9, 16, 23, 33, 42, 44, 54, 59, 67, 72]). Such choice was motivated by the fact that, first, as it is already argued in [62], structured P2P overlays provide a practical substrate for building scalable distributed applications due to the guarantees that they provide.

Secondly, the structural nature of the RDF data model influences the choice of the topology as it can be noticed through the data indexing model of most of the presented approaches.

Table 2. Qualitative comparison of P2P systems for RDF data storage and retrieval: s, p, o respectively refer to the subject, the predicate and the object of a RDF triple and c denotes the *context* of the application.

Topology		**Contribution details**			
		Data indexing	**Indexing particularities**	**Query language and Supported queries**	**Query processing**
Ring-based	RDFPeers [23] / MAAN [22]	Three-times `RDF triple` indexing.	Locality preserving hash function: `hash(s)`, `hash(p)`, `hash(o)`.	Atomic, conjunctive, disjunctive queries with the *same* subject	Recursive multi-predicate query resolution.
	Atlas [54] / Bamboo[76]	RDFPeers indexing scheme.	Hashing.	RQL; conjuctive triple pattern queries.	RDFPeers query processing algorithm; Subscriptions and matching mechanisms derive from QC and SBV [59].
	DSS [42] / Chord [83]	Semantic clustering.	Semantic-based RDF data organization.	RDQL.	Routing within the most suitable semantic cluster; flooding-based approach inside the cluster; Subscriptions mapping to a semantic cluster and forwarded to all nodes within the cluster, RDF mapping to the most suitable semantic cluster.
	PAGE [33] / Bamboo[76]	Six-times `quad` indexing.	`quads(s,p,o,c)` indexing instead of RDF triples; Associate an ID to each quad.	YARS Query Language.	Use `mask` and `hops` properties during the query resolution.
Cube-based	Edutella [71] / HyperCuP [80]	Semantic clustering at the Super-Peer level.	Super-Peer-Super-Peer (`SP-SP`) and Super-Peer-Peer (`SP-P`) indices.	RDF-QEL.	Lookup at the super-peer level based on `SP-SP` indices; Lookup inside the cluster based on `SP-P` indices.
	RDFCube [67] / CAN [75]	RDFPeers indexing scheme.	Store bit information about the existence of RDF triples.	Conjunctive queries.	Query mapping into a cell of RDFCube through hashing; Speed up *join queries* by performing an `AND` operation on bit information.
Tree-based	GridVine [9] / P-Grid [8]	Three-times RDF triple indexing at P-Grid layer; Schema indexing at the semantic mediation layer.	Order-preserving hashing; Semantic mediation layer overlay atop P-Grid.	OWL; atomic, conjunctive, disjunctive queries.	Schema mapping; Iterative query resolution.
	Unistore [49] / P-Grid [8]	Three-times indexing on the `OID`, the concatenation of $\mathtt{A_i}$ and $\mathtt{v_i}$, the value $\mathtt{v_i}$.	a tuple (`OID`, $\mathtt{v_1}$, ..., $\mathtt{v_n}$) of a given relation schema $\mathtt{R(A_1, \ldots, A_n)}$ is stored as `n` triples:(`OID`, $\mathtt{A_1}$, $\mathtt{v_1}$),..., (`OID`, $\mathtt{A_n}$, $\mathtt{v_n}$).	VQL; database and information retrieval-like queries (similarity, ranking, etc.).	Schema-independent query formulation, Semantic layer atop P-Grid [8]; Adaptive cost-aware query processing approach.
Generic-DHT-based	*Battré et al.* [16, 45]	RDFPeers-like indexing mechanism.	Different RDF repositories hosted by each node local/received/replica/generated repositories.	Not specified.	Knowledge and query graph-based models; Two-phase refinement procedure for query resolution; A look-ahead and Bloom filters enhancement techniques.
	QC and SBV [59]	Three-times (QC) and seven-times (SBV) RDF triple indexing.	QC: `hash(s)`, `hash(p)`, `hash(o)`; SVB: `hash(s)`,`hash(p)`, `hash(o)`, `hash(s+p)`, `hash(s+o)`, `hash(p+o)` `hash(s+p+o)`.	The query language is not clearly specified; atomic and conjunctive queries.	Query Chain(QC): Query evaluation by a *chain* of nodes and intermediate results flow through the nodes of the chain; last node in the chain delivers the result back to the query's originator; Spread by Value (SBV): construct multiple chains for the same query.

Several approaches [16, 23, 44, 54, 67], while based on different overlay topologies, share almost the same data indexing model by *hashing* the RDF triple elements (see Table 1) The main advantage of such indexing strategy is that triples with the same subject, object and predicate are stored on the same node and thus can be searched locally without needing to be collected from different data sources. However, nodes responsible for overly popular triples (e.g., `‘rdf:type’`, `‘dc:title’`), can be easily overloaded resulting in poor performances. One possible solution for this issue would be to use multiple hash functions to ensure a better load distribution as proposed in [75] and recently in [69].

The second category of RDF-based P2P approaches harnesses the semantic of RDF data either to build a "semantic" layer on top of the P2P overlay (e.g., GridVine [31]) or to adopt a semantic clustering approach and organize the P2P layer as function of the semantic of the stored data (e.g., Edutella [42, 72]). Others have extended the basic RDF data model by adding the "context" concept as in Page [33]. By taking into consideration data semantics in the data indexing phase, these approaches try to improve the data lookup. However, this needs an additional effort to maintain the mapping between the semantic and the overlay levels. Most of the presented works aim at adding data *availability* feature to the RDF storage infrastructure through *data replication* (e.g, [16, 23, 31, 54, 59]). However, data replication further raises several issues. Thereof, three main challenges have to be taken into consideration: which data items have to be replicated; where to place replicas and finally how to keep them consistent. In P2P systems there has been a lot of work on managing data replication. In [55], Ktari *et al.* investigated the performance of several replication strategies under DHTs systems including neighbor replication, path replication and multi-key replication. As argued in [55], the data replication strategy can have a significant impact on the system performance. Further effort may go into exploring more "dynamic" and adaptive replication approaches as function of data popularity or average peer online probability [52]. Although the replication techniques increase the data availability, they come not only at the expense of more storage space but can also affect the *data consistency* (e.g., concurrent update for the same triple). Moreover, the data inconsistency issue becomes even more intricate under the partitioning of the P2P network. Thus, an update operation might not address all replicas as a node storing a replica can be offline during the update process. Therefore, trade-offs are made between high data availability, data consistency and partition-tolerance. Brewer brought all these trade-offs together and presented the CAP theorem [40] which states that with `Consistency`, `Availability`, and `Partition-tolerance`, we can only ensure two out of these three properties. Recognizing which of the "CAP" properties the application really need is a fundamental step in building a successful distributed, scalable, highly reliable system.

Query processing. From the query processing point of view, sharing RDF data imposes new challenges on the distributed storage infrastructure related to supporting advanced query processing algorithms beyond simple keyword-based retrieval. Therefore, we need to take a deeper look at how the query can be optimized before being processed. The presented approaches for data retrieval and integration are mainly achieved either by fetching triples to query's originator which coordinates the query evaluation or forwarding intermediate results (e.g., [23, 59]) whenever the query is partially resolved. However, we believe that the first approach may not efficiently resolve conjunctive queries where each sub-query leads to a huge result set while the final join operation between them conducts to a small set. *Caching* results [14, 59] can alleviate this challenge, at least for similar queries, but can also affect the data consistency. Thereby, a trade-off is made between the network resource usage on one side and the information staleness on the other side. Other approaches such as in [14] decide at runtime whether the current result set has to be fetched to the query's originator or continue to be forwarded to other neighbors. As the query processing becomes more crucial especially when processing a huge data set such as the Billion Triple Challenge 2009 (BTC2009) dataset[3], some works have been proposed to reduce the large data sets to the interesting subsets as in [41]. To do so, BitMat [13], based on Bit Matrix conjunctive query execution approach, is used to generate compressed RDF structure. Therefore, a dominant challenge related to the distributed query processing [53] is how to improve the query performance and find an "optimal" query plan in order to enhance the query performance and reduce the communication cost. An already explored direction towards the query optimization plan is introduced by OptARQ [19], based on Jena ARQ [2], which uses the concept of *triple pattern selectivity estimation.* This concept makes use of statistical information about the ontological resources and evaluates the fraction of triples satisfying a given pattern. The goal is to find the query execution plan that minimizes the intermediate result set size of each triple pattern by join re-ordering strategies. Thereof, the smaller the selectivity the less intermediate results are likely to be produced. The presented works focusing on continuous queries [10, 24, 42, 54, 57, 58, 74], summarized in Table 3, have slightly different mechanisms for dealing with notifications and subscriptions. The only work based on a unstructured overlay network [10] relies on super-peers to manage and process subscriptions as well as route notifications while most of them are based on structured P2P overlays [24, 42, 54, 57, 58, 74] and directly take advantage of their underlying indexing mechanisms. Triples are indexed and retrieved using cryptographic functions (except for [74]). Ranger *et al.* [74] do not index data *a priori* but reuse the subscription management and group communication primitives used for notifications' propagation offered by Scribe [26]. By adding several caching mechanisms, in order to maintain the most popular queries and respective results fresh, they enhanced the data retrieval mechanism.

Pub/sub systems have been studied for some time now and are well established, but there exists a strong need for Quality of Service (QoS) [64], especially when deployed on top of a *best-effort* infrastructure such as the Internet. Now we will discuss some improvement opportunities with an emphasis on QoS. These improvements, even if they are valid for a lot of systems, were selected because

[3] http://vmlion25.deri.ie/

Table 3. Summary of the presented publish/subscribe systems for RDF data storage and retrieval

Pub/Sub System	Contribution highlights
Cai et al. [24]	– Subscription to atomic, disjunctive, range and conjunctive multi-predicate queries is possible. – Some combinations of conjunctive and disjunctive queries are not supported in the subscriptions. – A replication mechanism is provided for both subscriptions and data (for atomic queries), inherited from MAAN [22]. – Support for highly skewed subscription patterns is offered.
Chirita et al. [10]	– Super-peer-based system. – Formalism of the proposed system using a *typed first-order* language is given. – Resources advertisement managed by the super-peers using local partially order sets to reduce the network traffic. – Subscriptions are arranged in a hierarchical structure and take advantage of the *subscription subsumption* technique. – Notification matching and offline notifications' management are done by super-peers. – Integrated authentication mechanisms.
SQC and MQC [58]	– RDF queries in the style of RDQL. – Conjunctive multi-predicates queries are supported through two DHT-agnostic algorithms that extend the Query Chain (QC) algorithm [59] • *Single Query Chain* (SQC) • *Multiple Query Chain*(MQC)
CQC and CSBV [57]	– Only focuses on conjunctive queries (the motivation being that it is a core construct of RDF query languages). – Arbitrary continuous conjunctive queries are considered through two algorithms: • *Continuous Query Chain* (CQC): multiple nodes, forming a chain of fixed length, participate in the resolution of a subscription. • *Continuous Spread-By-Value* (CSBV): can be seen as a dynamic version of CQC where participating nodes are chosen during the subscription's processing. – Optimizations are provided to reduce network traffic, such as IP caching or query chain optimization.
Ranger et al. [74]	– Queries are expressed in a SPARQL dialect. – No data indexing *a priori* is used. – Various caching mechanisms (cached lookups, redundant caching, etc.) and grouping of related predicates of the same subject on the same node. – Queries are translated into a tree of atomic or complex queries. – Various logical trees of nodes are created during subscription processing, in which they can either *(i)* actively participate in a query resolution, *(ii)* forward query results to other nodes, *(iii)* simply act as consumers of query results.

few of presented pub/sub works actually focus on QoS. Enabling the Semantic Web at the Internet level means that we have no control on the architecture the application is deployed upon. Therefore, mechanisms dealing with this orthogonal issue should be taken into consideration.

Improvements related to subscription processing. Several techniques exist in order to improve the processing of subscriptions. Subscription *subsumption* [47] and *summarization* [85] are two optimization techniques whose goals are to reduce subscription dissemination traffic and enhance processing. The former exploits the "covering" relationship between a pair of subscriptions while the latter aims at representing them in a much more compact form. Well known pub/sub systems such as SIENA [25] and PADRES [56] implement *pair-wise* subscription cover checking to reduce redundancy in subscription forwarding. A more efficient approach, argued in [47], would be to exploit the subscription subsumption relationship between a new subscription and a set of existing active ones. In the presented pub/sub systems, only [10] takes advantage of this kind of subscription subsumption technique, while none uses summarization. We have yet to see the combination of these two improvement techniques applied to RDF-based pub/sub systems.

Improvements related to notification processing. Depending on the type of the application, there might be the need to ensure a (*stronger or weaker*) form of reliability as far as notifications are concerned. For instance, in a financial system such as the New York Stock Exchange (NYSE) or the the French Air Traffic Control System, reliability is *critical* as argued in [20]. Ensuring a correct dissemination of events despite failures requires a particular form of resiliency. As stated earlier, most of the presented works do not extensively offer QoS guarantees from the notification point of view (or even for subscription for that matter). Some of them use replication techniques to ensure data availability, but few if any discuss other aspects of QoS in depth such as the latency, bandwidth, delivery semantics, reliability and message ordering. As argued in [64], a lot of efforts are underway to build a generation of QoS-aware pub/sub systems. As a matter of fact, adaptation and QoS-awareness constitute major open research challenges that are naturally present in RDF-based pub/sub systems. The following three classes of service guarantees constitute some interesting challenges:

(i) Delivery semantics. Delivery semantics comes into play at the last hop, prior to notifying the subscriber. Notifications can be delivered *at most once*, *at least once*, *exactly once* or in a completely *best effort* way, that is with no guarantees of delivery whatsoever. All these delivery guarantees are implemented by a wide variety of broadcast algorithms which can be found in [43]. The message complexity in these algorithms is generally high and thus, depending also on the application guarantee requirements, they will have to be adapted to the overlay topology of the P2P system.
(ii) Reliability. The few works providing service guarantees generally follow a publisher-offered, subscriber (client)-requested pattern. None of the works take the reverse approach, that is, *providing application-specific quality of service guarantees explicitly specified by the client*, as argued in [63]. Earlier works at the overlay layer such as RON [11] and TAROM [84] focused their efforts on resiliency from the ground up, the problem is that they "only" consider link reliability without taking into account the node quality (e.g., load, subscribers/publishers degree, churn rate, etc.). Hermes [73] explicitly deals with routing robustness by introducing a reliability model at the routing level which enables event brokers to recover from failures. However, Hermes does not provide any client-specified service level reliability guarantees. Pub/sub systems which consider *event routing* based on reliability requirements are rare schemes as pointed out in [65].
(iii) Message ordering. In pub/sub systems, some applications, such as such as in financial systems and air traffic control emphasize on the preservation of a *temporal* order between the reception of events (e.g., FIFO/LIFO, priority, causal or total, etc.). Adopting a message ordering can have a significant impact not only on the routing algorithms but also on how the subscriber's node processes arriving messages. Strong dissemination abstractions such as a *Reliable Causal Broadcast*, well known and studied in the distributed systems literature, provide strong guarantees. However, because of their inherent inability to scale, their usage cost is prohibitive and limits their application in large-scale settings, as

argued in [43, 48]. Since the majority of the systems presented are built on top of structured overlay networks, one should take advantage of the overlay structure to disseminate events in a more efficient way, as in [26, 35]. Overall, there is a lot of room for improvements in pub/sub systems from a QoS point of view as pointed out, and the Semantic Web just might give us the opportunity to finally make these systems more reliable.

5 Conclusion

The Semantic Web enables users to model, manipulate and query information at a conceptual level in an unprecedented way. The underlying goal of this grand vision is to allow people to exchange information and link the conceptual layers of applications without falling into technicalities. At the core of the Semantic Web lies a powerful data model - RDF - an incarnation of the universal relation model developed in the 1980ies [66]. P2P technologies address *system complexity* by abandoning the centralization of knowledge and control in favor of a decentralized information storage and processing. The last generation of P2P networks - structured overlay networks [36] - represents an important step towards practical scalable systems. This survey presented various research efforts on RDF data indexing and retrieval in P2P systems. RDF-based P2P approaches discussed in this survey combine two research directions: research efforts that concentrate on the data model and query languages and efforts that attempt to take advantage of the expressiveness and the flexibility of such data model. The main objective was to build large scale distributed applications using the P2P technologies in order to come up with fully distributed and scalable infrastructures for RDF data storage and retrieval. As a natural extension, the publish/subscribe paradigm, built atop a P2P substrate, illustrates a more dynamic way of querying RDF data in a continuous manner. The main goal of these systems is to offer advanced query mechanisms and sophisticated information propagation among interested peers. Even though they are built on top of P2P systems, thus encompassing already discussed research directions found at the P2P level, they incorporate research issues of their own in terms of subscription processing and notification propagation. Leveraging orthogonal ideas, such as *federation*, could provide further control on the overall scalability and autonomy of the system by allowing different entities to keep the control on their own data while still collaborating. This structure inspired the work proposed by Baude *et al.* [17] to create the *Semantic Spaces*.

We have seen that the combination of P2P technologies such as structured overlay networks and the RDF data model is a promising step towards an effective Internet-scale Semantic Web. Both research communities made a tremendous effort, both on their own side, to provide solid and sound foundations. This interdisciplinary effort has proven that both communities can benefit from one another and that they can push forward the existing Web frontier.

References

[1] Gnutella RFC, `http://rfc-gnutella.sourceforge.net/`
[2] Jena - a Semantic Web Framework for java, `http://jena.sourceforge.net/`
[3] RDFStore, `http://rdfstore.sourceforge.net/`
[4] Resource Description Framework, `http://www.w3.org/RDF/`
[5] SETI@home, `http://setiathome.ssl.berkeley.edu/`
[6] SPARQL Query Language, `http://www.w3.org/TR/rdf-sparql-query/`
[7] W3C Semantic Web Activity, `http://www.w3.org/2001/sw/`
[8] Aberer, K., Cudré-Mauroux, P., Datta, A., Despotovic, Z., Hauswirth, M., Punceva, M., Schmidt, R.: P-Grid: a self-organizing structured P2P system. SIGMOD Record 32(3), 33 (2003)
[9] Aberer, K., Cudré-Mauroux, P., Hauswirth, M., Pelt, T.V.: GridVine: Building Internet-Scale Semantic Overlay Networks. In: International Semantic Web Conference (2004)
[10] Alex, P., Chirita, R., Idreos, S., Koubarakis, M., Nejdl, W.: Designing semantic publish/subscribe networks using super-peers. In: Semantic Web and Peer-To-Peer (January 2004)
[11] Andersen, D., Balakrishnan, H., Kaashoek, F., Morris, R.: Resilient overlay networks. In: Proceedings of the 18th ACM Symposium on Operating Systems Principles, pp. 131–145. ACM, Banff (2001)
[12] Antoniou, G., Harmelen, F.: Web ontology language: Owl. In: Handbook on Ontologies, pp. 91–110 (2009)
[13] Atre, M., Srinivasan, J., Hendler, J.: BitMat: A Main-memory Bit Matrix of RDF Triples for Conjunctive Triple Pattern Queries. In: 7th International Semantic Web Conference (ISWC) (October 2008)
[14] Battre, D.: Caching of intermediate results in DHT based RDF stores. Int. J. Metadata Semant. Ontologies 3(1), 84–93 (2008)
[15] Battré, D.: Query Planning in DHT Based RDF Stores. In: Proceedings of the 2008 IEEE International Conference on Signal Image Technology and Internet Based Systems (SITIS), pp. 187–194. IEEE Computer Society, Washington, DC, USA (2008)
[16] Battré, D., Heine, F., Höing, A., Kao, O.: On triple dissemination, forward-chaining, and load balancing in DHT based RDF stores. In: Moro, G., Bergamaschi, S., Joseph, S., Morin, J.-H., Ouksel, A.M. (eds.) DBISP2P 2005 and DBISP2P 2006. LNCS, vol. 4125, pp. 343–354. Springer, Heidelberg (2007)
[17] Baude, F., Filali, I., Huet, F., Legrand, V., Mathias, E., Merle, P., Ruz, C., Krummenacher, R., Simperl, E., Hamerling, C., Lorré, J.-P.: ESB Federation for Large-Scale SOA. In: Proceedings of the ACM Symposium on Applied Computing (SAC), pp. 2459–2466 (2010)
[18] Berners-Lee, T.: Linked data. W3C Design Issues (2006)
[19] Bernstein, A., Kiefer, C., Stocker, M.: OptARQ: A SPARQL Optimization Approach based on Triple Pattern Selectivity Estimation. Tech. rep., University of Zurich (2007)
[20] Birman, K.P.: A review of experiences with reliable multicast. Software: Practice and Experience 29(9), 741–774 (1999)
[21] Bloom, B.H.: Space/Time Trade-offs in Hash Coding With Allowable Errors. Commun. ACM 13(7), 422–426 (1970)
[22] Cai, M., Frank, M., Chen, J., Szekely, P.: MAAN: A multi-attribute Addressable Network for Grid Information Services. Journal of Grid Computing 2 (2003)

[23] Cai, M., Frank, M.R.: RDFPeers: a scalable distributed RDF Repository Based on a Structured Peer-to-Peer Network. In: WWW, pp. 650–657 (2004)

[24] Cai, M., Frank, M.R., Yan, B., MacGregor, R.M.: A subscribable Peer-to-Peer RDF Repository for Distributed Metadata Management. J. Web Sem. 2(2), 109–130 (2004)

[25] Carzaniga, A., Rosenblum, D.S., Wolf, A.L.: Design and Evaluation of a Wide-Area Event Notification Service. ACM Trans. Comput. Syst. 19(3), 332–383 (2001)

[26] Castro, M., Druschel, P., Kermarrec, A., Rowstron, A.: SCRIBE: A large-scale and decentralized application-level multicast infrastructure. IEEE Journal on Selected Areas in Communications 20(8), 1489–1499 (2002)

[27] Castro, M., Costa, M., Rowstron, A.: Debunking some myths about structured and unstructured overlays. In: Proceedings of the 2nd Conference on Symposium on Networked Systems Design and Implementation (NSDI), pp. 85–98. USENIX Association (2005)

[28] Magiridou, M., Sahtouris, S., Christophides, V., Koubarakis, M.: RUL: A declarative update language for RDF. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, pp. 506–521. Springer, Heidelberg (2005)

[29] Cohen, B., `http://www.bittorrent.com`

[30] Crespo, A., Garcia-Molina, H.: Semantic Overlay networks for P2P Systems. In: Agents and Peer-to-Peer Computing, pp. 1–13 (2005)

[31] Cudré-Mauroux, P., Agarwal, S., Aberer, K.: Gridvine: An infrastructure for peer information management. IEEE Internet Computing 11, 36–44 (2007)

[32] Dan Brickley, R.G.: RDF Vocabulary Description Language 1.0: RDF schema, `http://www.w3.org/TR/rdf-schema/`

[33] Della Valle, E., Turati, A., Ghioni, A.: *PAGE*: A distributed infrastructure for fostering RDF-based interoperability. In: Eliassen, F., Montresor, A. (eds.) DAIS 2006. LNCS, vol. 4025, pp. 347–353. Springer, Heidelberg (2006)

[34] Demers, A., Greene, D., Hauser, C., Irish, W., Larson, J., Shenker, S., Sturgis, H., Swinehart, D., Terry, D.: Epidemic algorithms for replicated database maintenance. In: Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing, pp. 1–12. ACM, Vancouver (1987)

[35] El-Ansary, S., Alima, L., Brand, P., Haridi, S.: Efficient broadcast in structured P2P networks. In: Peer-to-Peer Systems II, pp. 304–314. Springer, Heidelberg (2003)

[36] El-Ansary, S., Haridi, S.: An overview of structured overlay networks. In: Theoretical and Algorithmic Aspects of Sensor, Ad Hoc Wireless and Peer-to-Peer Networks. CRC Press, Boca Raton (2005)

[37] Eugster, P., Guerraoui, R., Kermarrec, A.M., Massoulie, L.: From Epidemics to Distributed Computing. IEEE Computer 37(5), 60–67 (2004)

[38] Eugster, P., Felber, P., Guerraoui, R., Kermarrec, A.: The many faces of publish/subscribe. ACM Computing Surveys (CSUR) 35(2), 114–131 (2003)

[39] Filali, I., Bongiovanni, F., Huet, F., Baude, F.: RDF Data Indexing and Retrieval: A survey of Peer-to-Peer based solutions. Research Report RR-7457, INRIA (November 2010)

[40] Gilbert, S., Lynch, N.: Brewer's Conjecture and the Feasibility of Consistent Available Partition-Tolerant Web Services. In: ACM SIGACT News, p. 2002 (2002)

[41] Williams, G.T., Weaver, J., Atre, M., Hendler, J.A.: Scalable Reduction of Large Datasets to Interesting Subsets. In: 8th International Semantic Web Conference (2009)

[42] Gu, T., Pung, H.K., Zhang, D.: Information Retrieval in Schema-based P2P Systems Using One-dimensional Semantic Space. Computer Networks 51(16), 4543–4560 (2007)

[43] Guerraoui, R., Rodrigues, L.: Introduction to reliable distributed programming. Springer-Verlag New York Inc., Secaucus (2006)

[44] Harth, A., Decker, S.: Optimized Index Structures for Querying RDF from the Web. In: Proceedings of the Third Latin American Web Congress (LA-WEB), p. 71. IEEE Computer Society, Washington, DC, USA (2005)

[45] Heine, F.: Scalable p2p based RDF querying. In: Proceedings of the 1st International Conference on Scalable Information Systems (InfoScale), p. 17. ACM, New York (2006)

[46] Broekstra, J.: Ehrig and Peter Haase and Frank van Harmelen and Maarten Menken and Peter Mika and Bjorn Schnizler and Ronny Siebes: Bibster - A Semantics-Based Bibliographic Peer-to-Peer System. In: The Second Workshop on Semantics in Peer-to-Peer and Grid Computing (SEMPGRID), New York (May 2004)

[47] Jafarpour, H., Hore, B., Mehrotra, S., Venkatasubramanian, N.: Subscription subsumption evaluation for content-based publish/Subscribe systems. In: Issarny, V., Schantz, R. (eds.) Middleware 2008. LNCS, vol. 5346, pp. 62–81. Springer, Heidelberg (2008)

[48] de Juan, R., Decker, H., Miedes, E., Armendariz, J.E., Munoz, F.D.: A Survey of Scalability Approaches for Reliable Causal Broadcasts. Tech. Rep. ITI-SIDI-2009/010 (2009)

[49] Karnstedt, M., Sattler, K.U., Richtarsky, M., Muller, J., Hauswirth, M., Schmidt, R., John, R., Ilmenau, T.U.: UniStore: querying a DHT-based universal storage. In: IEEE 23rd International Conference on Data Engineering (ICDE), pp. 1503–1504 (2007)

[50] Karvounarakis, G., Alexaki, S., Christophides, V., Plexousakis, D., Vouton, F.V., Scholl, M.: RQL: A Declarative Query Language for RDF. In: Proceedings of the 11th International Conference on World Wide Web (WWW), pp. 592–603. ACM Press, New York (2002)

[51] King, R.A., Hameurlain, A., Morvan, F.: Query Routing and Processing in Peer-To-Peer Data Sharing Systems. International Journal of Database Management Systems, 116–139 (2010)

[52] Knežević, P., Wombacher, A., Risse, T.: DHT-Based Self-adapting Replication Protocol for Achieving High Data Availability. In: Advanced Internet Based Systems and Applications, pp. 201–210 (2009)

[53] Kossmann, D.: The State of The Art in Distributed Query Processing. ACM Comput. Surv. 32(4), 422–469 (2000)

[54] Koubarakis, M., Miliaraki, I., Kaoudi, Z., Magiridou, M., Papadakis-Pesaresi, A.: Semantic Grid Resource Discovery using DHTs in Atlas. In: Proceedings of 3rd GGF Semantic Grid Workshop, Athens, Greece (February 2006)

[55] Ktari, S., Zoubert, M., Hecker, A., Labiod, H.: Performance Evaluation of Replication Strategies in DHTs Under Churn. In: Proceedings of the 6th International Conference on Mobile and Ubiquitous Multimedia (MUM), pp. 90–97. ACM, New York (2007)

[56] Li, G., Hou, S., Jacobsen, H.: A Unified Approach to Routing, Covering and Merging in Publish/Subscribe Systems Based on Modified Binary Decision Diagrams. In: Proceedings of 25th IEEE International Conference on Distributed Computing Systems (ICDCS) 2005, pp. 447–457 (2005)

[57] Liarou, E., Idreos, S., Koubarakis, M.: Continuous RDF query processing over dHTs. In: Aberer, K., Choi, K.-S., Noy, N., Allemang, D., Lee, K.-I., Nixon, L.J.B., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P. (eds.) ASWC 2007 and ISWC 2007. LNCS, vol. 4825, pp. 324–339. Springer, Heidelberg (2007)
[58] Liarou, E., Idreos, S., Koubarakis, M.: Publish/Subscribe with RDF data over large structured overlay networks. In: Moro, G., Bergamaschi, S., Joseph, S., Morin, J.-H., Ouksel, A.M. (eds.) DBISP2P 2005 and DBISP2P 2006. LNCS, vol. 4125, pp. 135–146. Springer, Heidelberg (2007)
[59] Liarou, E., Idreos, S., Koubarakis, M.: Evaluating conjunctive triple pattern queries over large structured overlay networks. In: Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L.M. (eds.) ISWC 2006. LNCS, vol. 4273, pp. 399–413. Springer, Heidelberg (2006)
[60] Liu, Y., Plale, B.: Survey of Publish Subscribe Event Systems. Tech. rep., Indiana University (2003)
[61] Loo, B.T., Huebsch, R., Stoica, I., Hellerstein, J.M.: The Case for a Hybrid P2P Search Infrastructure. In: Peer-to-Peer Systems III, pp. 141–150 (2005)
[62] Lua, K., Crowcroft, J., Pias, M., Sharma, R., Lim, S.: A Survey and Comparison of Peer-to-Peer Overlay Network Schemes. IEEE Communications Surveys and Tutorials, 72–93 (2005)
[63] Mahambre, S.P., Bellur, U.: An Adaptive Approach for Ensuring Reliability in Event Based Middleware. In: Proceedings of the Second International Conference on Distributed Event-based Systems, pp. 157–168. ACM, New York (2008)
[64] Mahambre, S.P., Madhu Kumar, S.D., Bellur, U.: A Taxonomy of QoS-Aware, Adaptive Event-Dissemination Middleware. IEEE Internet Computing 11(4), 35–44 (2007)
[65] Mahambre, S., Bellur, U.: Reliable Routing of Event Notifications over P2P Overlay Routing Substrate in Event Based Middleware. In: IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 1–8 (2007)
[66] Maier, D., Ullman, J.D., Vardi, M.Y.: On the Foundations of the Universal Relation Model. ACM Trans. Database Syst. 9(2), 283–308 (1984)
[67] Matono, A., Pahlevi, S., Kojima, I.: RDFCube: A P2P-Based Three-Dimensional Index for Structural Joins on Distributed Triple Stores. In: Databases, Information Systems, and Peer-to-Peer Computing, pp. 323–330 (2007)
[68] Meshkova, E., Riihijärvi, J., Petrova, M., Mähönen, P.: A Survey on Resource Discovery Mechanisms, Peer-to-Peer and Service Discovery Frameworks. Comput. Netw. 52(11), 2097–2128 (2008)
[69] Mu, Y., Yu, C., Ma, T., Zhang, C., Zheng, W., Zhang, X.: Dynamic Load Balancing With Multiple Hash Functions in Structured P2P Systems. In: Proceedings of the 5th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM), pp. 5364–5367. IEEE Press, Piscataway (2009)
[70] Mühl, G., Fiege, L., Pietzuch, P.: Distributed Event-Based Systems. Springer-Verlag New York, Inc., Secaucus (2006)
[71] Nejdl, W., Wolf, B., Qu, C., Decker, S., Sintek, M., Naeve, A., Nilsson, M., Palmer, M., Risch, T.: Edutella: A P2P Networking Infrastructure Based on RDF. In: Proceedings of the 11 International World Wide Web Conference, WWW (May 2002)
[72] Nejdl, W., Wolpers, M., Siberski, W., Schmitz, C., Schlosser, M., Brunkhorst, I., Löser, A.: Super-peer-based routing and clustering strategies for RDF-based peer-to-peer networks. In: Proceedings of the 12th International Conference on World Wide Web, pp. 536–543. ACM, New York (2003)

[73] Pietzuch, P., Bacon, J.: Hermes: A distributed Event-based Middleware Architecture. In: Proceedings of the 22nd International Conference on Distributed Computing Systems(ICDCS), pp. 611–618 (2002)

[74] Ranger, D., Cloutier, J.F.: Scalable Peer-to-Peer RDF Query Algorithm. In: Proceedings of Web information systems engineering International Workshops (WISE), p. 266 (November 2005)

[75] Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A Scalable Content-Addressable Network. In: Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM), pp. 161–172. ACM, New York (2001)

[76] Rhea, S., Geels, D., Roscoe, T., Kubiatowicz, J.: Handling Churn in a DHT. In: Proceedings of the Annual Conference on USENIX Annual Technical Conference (ATEC), p. 10 (2004)

[77] Risson, J., Moors, T.: Survey of Research Towards Robust Peer-to-Peer Networks: Search Methods. Computer Networks 50(17), 3485–3521 (2006)

[78] Rowstron, A., Druschel, P.: Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: Liu, H. (ed.) Middleware 2001. LNCS, vol. 2218, pp. 329–350. Springer, Heidelberg (2001)

[79] Guha, R.V.: rdfDB: An RDF Database, `http://guha.com/rdfdb/`

[80] Schlosser, M.T., Sintek, M., Decker, S., Nejdl, W.: HyperCuP - Hypercubes, Ontologies, and Efficient Search on Peer-to-Peer Networks. In: Moro, G., Koubarakis, M. (eds.) AP2PC 2002. LNCS (LNAI), vol. 2530, pp. 112–124. Springer, Heidelberg (2003)

[81] Seaborne, A.: RDQL - A Query Language for RDF. Tech. rep., W3C, proposal (2004)

[82] Staab, S., Stuckenschmidt, H.: Semantic Web and Peer-to-Peer. Springer, Heidelberg (2006)

[83] Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In: Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM), pp. 149–160. ACM, New York (2001)

[84] Tang, C., McKinley, P.: Improving Multipath Reliability in Topology-Aware Overlay Networks. In: proceedings of the 25th IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW), pp. 82–88. IEEE, Los Alamitos (2005)

[85] Triantafillou, P., Economides, A.: Subscription Summarization: A New Paradigm for Efficient Publish/Subscribe Systems. In: Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS 2004), pp. 562–571. IEEE Computer Society, Los Alamitos (2004)

[86] Voulgaris, S., Rivière, E., Kermarrec, A.M., Steen, M.V.: Sub-2-sub: Self-organizing content-based publish subscribe for dynamic large scale collaborative networks. In: Proceedings of the fifth International Workshop on Peer-to-Peer Systems, IPTPS (2006)

[87] Yang, B., Garcia-Molina, H.: Designing a Super-Peer Network. In: Proceedings of the 19th International Conference on Data Engineering (ICDE), vol. 1063, p. 17 (2003)

[88] Zhou, J., Hall, W., Roure, D.D.: Building a Distributed Infrastructure for Scalable Triple Stores. Journal of Computer Science and Technology 24(3), 447–462 (2009)