HAL
archives-ouvertes.fr

RDF Data Indexing and Retrieval: A survey of Peer-to-Peer based solutions

Imen Filali, Francesco Bongiovanni, Fabrice Huet, Françoise Baude

▶ **To cite this version:**

Imen Filali, Francesco Bongiovanni, Fabrice Huet, Françoise Baude. RDF Data Indexing and Retrieval: A survey of Peer-to-Peer based solutions. [Research Report] RR-7457, INRIA. 2010. <inria-00540314>

HAL Id: inria-00540314
https://hal.inria.fr/inria-00540314

Submitted on 26 Nov 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

RDF Data Indexing and Retrieval: A survey of Peer-to-Peer based solutions

Imen Filali — Francesco Bongiovanni — Fabrice Huet — Françoise Baude

N° 7457

Novembre 2010

Distributed Systems and Services

ISSN 0249-6399 ISRN INRIA/RR--7457--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
SOPHIA ANTIPOLIS - MÉDITERRANÉE

RDF Data Indexing and Retrieval: A survey of Peer-to-Peer based solutions

Imen Filali, Francesco Bongiovanni, Fabrice Huet, Françoise Baude

Theme : Distributed Systems and Services
Networks, Systems and Services, Distributed Computing
Équipe-Projet Oasis

Rapport de recherche n° 7457 — Novembre 2010 — 54 pages

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 and support various types of queries. The attractive features of the Peer-to-Peer (P2P) communication model 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 that adopt this combination and gives an insight on how to deal with the RDF data at the indexing and querying levels.

Key-words: 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.

Centre de recherche INRIA Sophia Antipolis – Méditerranée
2004, route des Lucioles, BP 93, 06902 Sophia Antipolis Cedex
Téléphone : +33 4 92 38 77 77 — Télécopie : +33 4 92 38 77 65

Indexation et stockage de données RDF: une étude de solutions basées sur les systèmes pair-à-pair

Résumé : Le Web Sémantique permet de modéliser, créer et faire des requêtes sur les ressources disponibles sur le Web. Afin de permettre à ses technologies d'exploiter leurs potentiels à l'échelle de l'Internet, il est nécessaire qu'elles reposent sur des infrastructures qui puissent passer à l'échelle ainsi que de répondre aux exigences d'expressivité des types de requêtes qu'elles offrent.

Les bonnes propriétés qu'offrent les dernières générations de systèmes pair-à-pair en termes de décentralisation, de tolérance aux pannes ainsi que de passage à l'échelle en font d'eux des candidats prometteurs. La combinaison du modèle pair-à-pair et des technologies du Web Sémantique est une tentative innovante ayant pour but de fournir une infrastructure capable de passer à l'échelle et pouvant stocker et rechercher des données de type RDF.

Dans ce contexte, ce rapport présente un état de l'art et discute en détail des travaux autour de systèmes pair-à-pair qui traitent des données de type RDF à large échelle. Nous détaillons leurs mécanismes d'indexation de données ainsi que le traitement des divers types de requêtes offerts.

Mots-clés : Web Sémantique, pair-à-pair (P2P), Table de Hashage Distribuée, Resource Description Framework (RDF), dépot distribué, indexation de données RDF, traitement de requêtes RDF, publication/souscription, traitement de souscriptions.

Contents

1 Introduction

Pushed by the willingness to realize the Semantic Web [7] and more specifically the "Linked Data" vision [20] of Sir Tim Berners Lee at the Internet level, the last few years have 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], allowing a flexible description of both data and metadata in large scale settings. This combination arose quite naturally, and, as stated in [113], both address the same need but at different levels. The Semantic Web allows 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 information using the decentralized organization principle. P2P systems have been recognized as a key communication model to build scalable platforms for distributed applications such as file sharing (e.g., Gnutella [1]) 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 [122]. *Structured* P2P overlays [36] emerged to alleviate inherent problems of unstructured overlays. In these systems, peers are organized in a well-defined geometric topology (e.g., ring, torus, etc.) and, compared to unstructured networks, they exhibit stronger guarantees in terms of search time and nodes' maintenance [37]. By providing a *Distributed Hash Table (DHT)* abstraction, by the mean of *Put* and *Get* methods, they offer simple mechanisms to store and fetch data easily 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, 67, 112]) 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 and 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 focus on P2P networks (e.g., [69, 92, 85, 103]) or the pub/sub communication model (e.g., [49, 83]) 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 [85] 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 [103, 92] but even if they have a wider scope than [85], as they cover several service discovery frameworks, search methods,..., 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, this survey introduces in Section 2 the basic terminologies and technologies that will be used while investigating RDF-based P2P systems. This includes the main concepts behind the RDF data model as well as set of canonical structured P2P networks. 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. It also includes complementary techniques used to improve the query processing through many strategies such as parallel query processing, caching, replication, etc. Section 4 investigates the publish/subscribe communication model on top of RDF-based P2P systems and underlines various algorithms that have been proposed to manage complex subscriptions. General discussions and challenges are laid out for both Sections 3 and 4 before concluding the survey in Section 5 with a short summary and open perspectives.

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]. Processing a huge amount of information at the Web scale, from an architectural point of view, is not a trivial task. 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 ever-growing need of up-to-date data requires more advanced algorithms and techniques in order to satisfy this requirement. 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 *(s, p, 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. Triples form a directed graph where arcs are always directed from resources (subjects) to values (objects). Figure 1 gives a simplified graph presentation of the triple *(book, hasTitle, Semantic Web)*. The resource nodes have respectively *book* and *Semantic Web* as values.

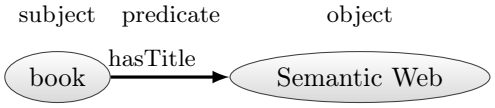

Figure 1: Example of an RDF graph presenting an RDF statement.

RDFS [41] 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 [31] atomic triple queries can be summarized into eight queries patterns: $(?s, ?p, ?o)$, $(?s, ?p, o_i)$,$(?s, p_i, ?o)$, $(?s_i, ?p_i, o_i)$, $(s_i, ?p, ?o)$, $(s_i, ?p, o_i)$, $(?s_i, p_i, o)$, (s_i, p_i, o_i).[1]. In each query pattern, $?x$ denotes the element(s) that the query is looking for and x_i refers to a constant value of the RDF element. For instance, the query pattern $(?s, p_i, o_i)$ returns all subjects s for a given predicate p_i and object o_i (if they exists).

[1]From now on, the triple (s_i, p_i, o_i) denotes the RDF triple elements: the subject s_i, the predicate p_i, the object o_i.

- **Conjunctive queries** are expressed as a conjunction of a list of atomic triple patterns (sub-queries). For instance, a conjunctive query may look like: $(?s_1, p_1, o_1) \wedge (?s_2, p_2, o_2)$.
- **Range queries** involve queries for single or multiple attribute data whose attribute value falls into a range defined by the query. As an example $q = (s, p, ?o)$ `FILTER` $o < v$, looks, for a given subject s and a predicate p, for all object values bounded by the value 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 Structured P2P Systems

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 [37]. P2P overlays such as CAN [101], Chord [114], Pastry [104], 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.

2.2.1 Chord

In Chord [114], proposed by stoica *et al.*, peers are organized in a logical ring topology. Each Chord 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 $m = log(N)$ routing entries, called *fingers*. A $finger[i]$ is equals to $successor(n \oplus 2^{i-1})$ with $1 \leq i \leq m$, where $(n \oplus 2^{i-1})$ is $(n + 2^{i-1})$ *modulo* N. Figure 2 shows an example of Chord overlay with 11 nodes as well as the finger table of node n_8. Chord uses consistent hashing [64] to ensure, with a high probability, a load balancing of keys among peers as they roughly receive the same number of keys. To perform a $lookup(k)$, the query is propagated along the ring in the clockwise direction where, upon receipt, each node will forward it to the closest node identifier in its finger table

that precedes or equals the key k. The lookup routing requires $O(log(N))$ (worst case[2]) messages.

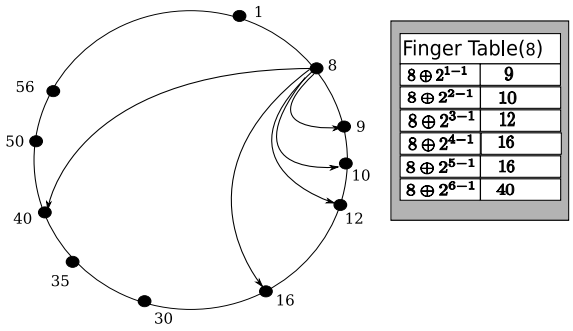

Figure 2: Finger table of node 8 in a Chord overlay of 11 nodes in a 6-bit identifier space. Finger $[i]$ is the first node that succeeds $(8 \oplus 2^{i-1})$.

2.2.2 Bamboo

Bamboo [102], as Chord, builds a one-dimensional circular identifier space. Bamboo uses Pastry [104] as basis for its topology and routing algorithms while improving its resilience under churn. More precisely, peers identifiers are treated as a sequence of digits of base 2^b. A node, say n, maintains two sets of neighbors: the *leaf set* and the *routing table*. The leaf set, denoted L, refers to the set of $2k$ nodes that immediately precede and follow n in the circular identifier space. The routing table includes the set of nodes whose identifiers share the successively longer prefixes with n. Assume that l is the row in the peer routing table, the routing table entry at the column i and row l, denoted by $R_l[i]$, stores the neighbor whose identifier matches the peer n identifier in exactly l digits and whose $(l+1)^{th}$ digit is i. Figure 3 gives an example of the routing table leaf set of the node identified by 10233102. Note that, unlike Chord, Bamboo uses prefix routing mechanism to route message to the peer responsible for the searched key, that is, in each routing step, a node forwards the message to the peer that shares at least one digit longer than the prefix that the key shares with the current node's *id*. If such node is unknown by the current node, the message will then be send to the node whose *id* shares a prefix with the key as long as the current node but it is numerically closer to the key than the current node's id. Let us denote by $K[i]$ the i^{th} digit of the key K. If $R_l[K[l]]$ is not empty, the message will be forwarded to that node. Otherwise, it will be sent to the node n in L such as n is the numerically closest to K. Data item identified by id' is stored at the peer with the closest numerical *id* to id'. This differs from the Chord approach where data are stored at the first peer with *id* equal or greater than id'. As in Chord, Bamboo performs a lookup in $O(log(N))$ hops, with N being the maximum number of nodes. Adding the leaf set pointers improve the overall resilience as they ensure, to a certain extent, the overlay's connectivity especially under node failure.

[2]From here on, whenever a complexity (query, lookup,etc.) is specified, it will be the *worst case* one.

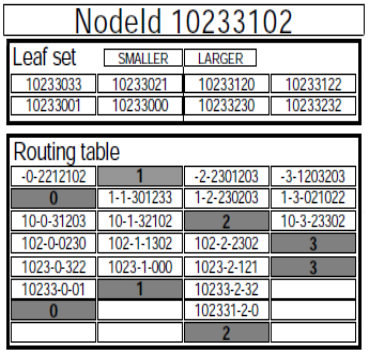

Figure 3: The leaf set and the routing table in Bamboo (and Pastry) node with the identifier 10233102, $b = 2$, $k = 2$. Digits are in base 4. The shaded cell in each row of the routing table refers to the corresponding digit of the current node id 10233102. The nodeId at each cell shows the *common prefix with* 10233102-*next digit-rest of nodeId* (from [104]).

2.2.3 Content Addressable Network (CAN)

The Content Addressable Network (CAN) [101] is a structured P2P network based on a d-dimensional Cartesian coordinate space labeled $\mathcal{D}$. This space is dynamically partitioned among all peers in the system such that each node "owns" a zone in $\mathcal{D}$ and uses it to store *(key, value)* pairs. For instance, to store the (k, v) pair, the key k is deterministically mapped onto a point i in $\mathcal{D}$ and then the value v is stored at the owner of the zone comprising i. The *lookup*(k) corresponding to a key k is achieved by applying the same deterministic function on k to map it onto i. A query for a given value will be routed through intermediate nodes in the overlay until it reaches the peer's zone containing i. When a peer joins the CAN overlay, it picks a random point p belonging to $\mathcal{D}$ and a *JOIN_QUERY* message will be routed to the zone that contains that point. A zone will be then allocated to the new peer by splitting the current owner's zone in half: keeping half for the original peer owner and allocate the other one to the new peer. Figure 4 depicts a two dimensional CAN space before and after the joining of node E.
Other variations of the CAN overlay such as [26, 27, 107] improve a set of CAN's features as for example the query response time or the load balancing. In a CAN Cartesian space with d dimensions and N peers, each node has to maintain $O(d)$ routing information while lookup complexity is $O(dN^{\frac{1}{d}})$.
Although these systems (i.e., Chord, Bamboo, CAN) show good performance and scalability characteristics, they only support *exact queries*, i.e., querying for data items matching a given key. Moreover, the hashing mechanism used in such approaches works well with static object identifiers such as file names, but it is not suitable to handle dynamic object content. In particular, the ability to perform more complex queries over changeable datastores is a key feature of many systems such as distributed databases. A enormous research effort has been achieved in this context and led to a set of fully distributed P2P approaches such as [30, 8] having the capabilities to manage more complex queries

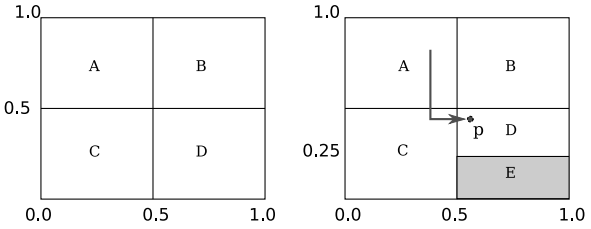

Figure 4: Example of 2-dimensional CAN overlay before and after the joining of node E.

such as range queries [50]. The remainder of this section focuses on such P2P approaches.

2.2.4 P-Grid

P-Grid [8] is a peer-to-peer lookup system based on a virtual distributed binary tree. Each peer $n \in N$ is associated with a leaf node of the tree. Each leaf corresponds to a binary string $\pi \in \Pi$ called the *key-space partition* where Π is the entire key partition. As a result, each P-Grid node is associated with a path $\pi(n)$ and its position in the overlay is determined by its path. This position indicates the subset of the tree's overall information that it is responsible for. Thus, each peer is responsible for storing keys that fall under its current key space[3]. Every peer n, labeled $\pi(n)$, maintains a set of references to peers sharing a common prefix of different sizes with $\pi(n)$. Formally, for each prefix, $\pi(n,l)$ for $\pi(n)$ with length l such as $0 < l < \pi$, n maintains a pointer to peer q satisfying the property $\overline{\pi(n,l)} = \pi(q,l)$, where $\overline{\pi}$ is the binary string π with the last bit inverted. As a result, the cost for storing these references is bounded by the depth of the tree. Keys are generated using an order preserving hash function, i.e., for two strings s_1 and s_2: $s_1 \subseteq s_2 \Rightarrow key(s_1) \subseteq key(s_2)$. This property allows P-Grid to efficiently manage range queries. Datta *et al.* propose in [42] two algorithms to manage such kind of queries: the *min-max traversal* algorithm and the *shower* algorithm. In the min-max travel algorithm, queries are processed sequentially by starting from the peer that manages data belonging to the upper or the lower bound of the range. Queries will then be forwarded to the peer responsible for the next key partition until the peer responsible of the other bound of the query is found. In the shower algorithm, queries are processed concurrently. The range query is first forwarded to an arbitrary peer responsible of any key space partition that falls into the query's range. From there, the query will be forwarded to other partitions within the range.
An example of a P-Grid tree is given by Figure 5. The routing table entry is in the form "p:x" and means that keys with prefix p will be routed to peer x. Queries in P-Grid are resolved by prefix matching. An example of query routing is also given by Figure 5 where the peer 1 receives a query for a key "110". As it is only reponsible for data items whose keys starting by "00", it checks in its routing table for the "closer" node to the result. Therefore, it forwards the query to the peer 3 whose path starts by 1. Similary, once the query is received by the peer 3, it checks if it can satisfy the query. Since it only stores keys starting

[3]e.g., peer's path is the prefix of the key of each stored data item.

by "10", it forwards the query to node 4 where the prefix "110" falls into its key partition space. To manage fault tolerance and query load balancing, multiple

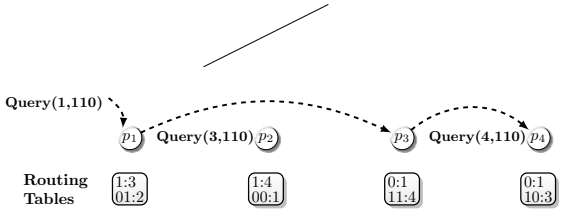

Figure 5: Example of P-Grid binary tree.

peers can be associated with the same key partition. Epidemic algorithms [82] are used to keep the replicated content up-to-date.

2.2.5 MAAN

Multi-Attribute Addressable Network (MAAN) proposed in [30] is considered as an extension of Chord which handles multi-attribute range queries. Items are mapped to the Chord m-bit key space using a SHA-1 hash function for string value attributes and a locality preserving hash function H for numerical values. Each resource attribute has its owns registration, and each registration is composed by a pair *<attribute-value, resource-info>*. Every node is responsible for storing keys that fall into its key space. In MAAN, the evaluation of multi-attribute range queries is implemented in two different ways: *iterative* and *single attribute dominated routing*. In the *iterative-based* approach, for a query composed of M sub-queries, each sub-query is resolved separately in its proper attribute space. Results are then collected and intersected at the query originator. The search complexity in terms of routing hops is $O(\sum_{i=1}^{M}(logN + N \cdot s_i))$ where M is the number of sub-queries, N the number of peers and s_i the *selectivity*[4] of the sub-query i. In the *single attribute dominated routing*, the number of routing hops needed to resolve the query is reduced as search results of a query must satisfy all the sub-queries on each attribute dimension and the intersection set of all resources that satisfies each sub-query. In other words, assume that X is the set of resources satisfying the query Q, then X should satisfy all the sub-queries of Q. Therefore, we have $X = \cap_{1 \leq i \leq M} X_i$, where X_i is the set that satisfies the sub-query on attribute a_i. The search request in this algorithm looks like $Search\_Request(k, a, R, O, X)$ where k is key used for routing[5], a is the dominated attribute[6], R is the desired attribute value range $[l, u]$, O is the list of sub-queries of all other attributes except a and finally, X is the set of the discovered resources satisfying all sub-queries. When a peer n performs a search request, it first routes the query to the node n_l such as $n_l = successor(H(l))$. The node n_l looks in its local index for resources where attribute values fall

[4]The *selectivity* is defined as the ratio of the query range width in identifier space to the size of the whole identifier space.

[5]Initially, $k = H(l)$ such as H is the hash function and l is the lower bound of the attribute value range.

[6]The dominated attribute is the attribute with the lowest selectivity.

P2P system	Architecture	Data indexing	Queries: type and evaluation	Lookup complexity
Chord [114]	ring-based	map *(key, value)* to the node with the closest identifier to *key*	• key-based queries	$\mathcal{O}(log(N))$
Bamboo [102]	ring-based	matching key to the *node identifier* maintaining leaf set and routing table	• key-based queries • Route query to the node with the closest numerical number to the key	$\mathcal{O}(log(N))$
CAN [101]	*d*-dimensional Coordinate space	map *(key, value)* in the Coordinate space	• key-based queries	$\mathcal{O}(dN^{\frac{1}{d}})$
P-Grid [8]	binary tree	matching a prefix to the *node identifier*	• key-based and range queries	$\mathcal{O}(log(\pi))$
MAAN [30]	Chord-based	Chord based data indexing scheme	• key-based and multi-attribute range queries • Iterative based: each sub-query resolved separately (^{a}) • Single attribute dominated routing: results satisfying all the sub-queries for each attribute (^{b})	(^{a}) $\mathcal{O}(\sum_{i=1}^{M}(logN + N \cdot s_i))$ (^{b}) $\mathcal{O}(logN + N \cdot S_{min})$

Table 1: Structured P2P approaches- N: network size, d: number of dimensions, π: key-space partition, s_i: sub-query selectivity, S_{min}: minimum selectivity, M: number of sub-queries.

into $[l, v]$ and for attribute values satisfying all other sub-queries in Q. If such results are found, they will be appended to X. This procedure is repeated until the search request is received by $n_u = successor(H(u))$. The node n_u will then send the result set X to the query's initiator. As this approach needs to perform only one iteration for the dominated attribute , this method has a complexity of $O(logN + N \cdot S_{min})$, where S_{min} is the minimum selectivity of all attributes.

Table 1 summarizes the basic characteristics of the structured P2P approaches discussed in this section. It includes the overlay architecture, the data indexing model, the supported queries and the lookup complexity.

2.3 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., structured overlays and adding a standard abstraction on top of it, i.e., DHT. The first wave of DHTs [114, 101] 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 DHTs led to a set of P2P approaches such as [8, 30] having the capabilities to manage more complex queries such as range queries [8] or multi-attribute queries [30]. 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, three 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 ?

Data retrieval How can we take advantage of P2P search mechanisms to efficiently query and retrieve RDF data in large scale settings ? Moreover, what is the impact of the data indexing methods on the query processing algorithms ?

Data integration [7] As a query, composed by a set of sub-queries, can be answered by several nodes, how to efficiently combine RDF data residing in different locations and provide a unified view of those data ?

3 RDF Data Storage and Retrieval in P2P Systems

Centralized RDF repositories and lookup systems such as RDFStore [3], Jena [2], RDFDB [106] 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 (e.g., bottlenecks, unscalable, poor failure handling, etc.). 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. However, data distribution across the network comes at the expense of the complexity of the query processing phase. Accordingly, the greatest challenges faced by these systems are the data organization and query processing, because queries over RDF data are typically complex and, if not carefully optimized, may induce low search performance. Therefore, algorithms are needed to efficiently[8] process RDF queries in such decentralized settings. The remainder of this section presents a selection of research works combining the RDF data model with a P2P architecture, with a focus on data indexing and processing in structured P2P networks.

3.1 RDF Data Storage and Retrieval in Unstructured P2P Networks

3.1.1 Bibster

Bibster [61] is a semantic-based P2P system for sharing bibliographic metadata. Bibster nodes build an unstructured P2P network and use JXTA [53] as the

[7] Generally, this operation is combined with the data retrieval phase.
[8] E.g., efficiency in terms of query processing, load distribution, etc.

communication platform.
Data indexing. Each Bibster node manages a *Local Node Repository* which is based on Sesame RDF-S repository [29] and stores the bibliographic metadata. Sesame RDF Query Language (SeRQL) [28] is used for data querying and query reformulation. In Bibster, peers share an ontology which is used to describe both the expertise of each node and the subject of query. A "*peer's expertise*" is defined as the semantic description of the local node repository. The query subject in this context is the abstraction of a query q. It is expressed in terms of the common ontologies and specifies the required expertise to answer that query. Based on the exchange of these expertises between peers, a semantic topology is built on top of the P2P architecture.
Query processing. Upon receiving a query, a peer tries to evaluate it against its local repository. If no answer is available, it runs a *peer selection* algorithm in order to forward the query only to peers that seem to be able to answer the query. This algorithm uses a *similarity function* to compute the similarity between subject of the query and the advertised expertise topics. The similarity function in Bibster, defined as Sim_{Topics}, is based on the idea that topics which are close in terms of their position in the topic hierarchy are more similar than topics that have a large distance.

3.1.2 S-RDF

In [125], Zhou *et al.* proposed a distributed RDF infrastructure based on an unstructured P2P topology called S-RDF.
Data indexing. In this approach, a single RDF repository is split into several RDF files. RDF data are organized on *semantic aware* hierarchy where RDF triples sharing the same subject are assigned to the same RDF file. Each peer in S-RDF system is responsible for maintaining and publishing RDF triples. RDF triples, locally stored, are partitioned as function of their subjects. Upon joining the network, a peer has to advertise its local RDF data and builds its own *neighbor table*. The neighbor table of a peer p stores relevant information about each neighbor n_i. This information includes the identifier of the neighbor n_i, its descriptive terms and the number of *semantically-related* descriptive terms respectively managed by n_i and p. To keep the neighbor table up-to-date, peers are periodically probing their neighbors. When a peer p leaves the network, its neighbors have to update their tables by removing entries associated to p.
Query processing. The proposed S-RDF architecture supports ontology-based matching. The matching process is performed either between a received query and the descriptive terms of the current peer or between descriptive terms of two different peers. It uses a similarity function to compute similarity between two data items. A match is found if a descriptive term is *semantically similar* to at least one descriptive term of the peer. A *semantic directed search* (SDS) protocol, inspired from random walk [86] forwarding mechanism, is used for query routing. More precisely, the query routing process takes advantage of the semantic relation between triples and queries (via a similarity function) to forward queries to the most relevant peers. If no significantt similarity is found during the matching process, the query is forwarded to all neighbors. At each forwarding step, a copy of the query is sent to the neighbor with the highest potentiality, after being tagged by *"COP (Continue tO Propagated)"*. If only a set $\mathcal{S}$ of peer p's neighbors have a "good" potentiality, a peer $n \in \mathcal{S}$ is selected

to forward a query's copy labeled *COP*. For all others neighbors, a copy of the query, denoted by a *NO_ COP* tag will be sent for each neighbor n_j such as $n_j \notin \mathcal{S}$. Query will be evaluated locally after fetching results from peers answering it to the query 's originator. Note that since RDF triples are assigned to files according to their subjects, queries in the form of *(?s,p,o)* can either be flooded in the network or routed using other techniques such as expanding ring[9]. When issuing more complex queries, users have to provide information about "potential subjects" that will help narrowing down the data files. If such information is missing, a flooding-based mechanism is used to locate the requested triples. While the authors indicate that a conjunctive query may be parsed to several sub-queries through the query processor engine, they do not provide details on how this procedure could be accomplished (sub-query dependencies).

3.2 RDF Data Storage and Retrieval in Structured P2P Networks

In this section, we focus on distributed RDF repositories that use structured P2P systems. Even if most of them use cryptographic algorithms for data indexing, they notably differ in query processing and data integration mechanisms.

3.2.1 Edutella

In a subsequent work of Edutella [97] RDF-based P2P infrastructure, Nejdl *et al.* propose a super-peer based architecture [98]. In this 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 super-peers. The super-peers are connected in a so-called *HyperCuP* topology [109] and are responsible for 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. Edutella implements a schema-based P2P system. In other words, the system is aware of schema information and take it into consideration for query optimization and routing. Each peer sends indices of its data to its super-peer. This information can be stored in different granularities, like schema, property, property value or property value range. However, the index does not contain individual data entries but indexes peers. This kind of index is called a *Super-Peer-Peer (SP-P)* index. For example, a super-peer receiving a query with a given schema, will forward it only to peers that support 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. As for the SP-P index, this strategy can reduce the traffic overhead between super-peers in the Edutella network.

Data indexing. When a new peer, say p, registers to a super-peer, labeled sp, it advertises the necessary schema information to sp. The super-peer 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

[9]The principle behind it is to issue the query with a small TTL value. If no result is found, resend the same query with a larger TTL.

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 message broadcasting during the query routing phase. The similarity-based clustering approach tries to integrate new peers with existing peers that have similar characteristics based, for example, on a topic ontology shared by peers. Edutella supports the RDF query exchange language (RDF-QEL).

3.2.2 RDFPeers

RDFPeers [31] was the first P2P system to propose the use of DHTs in order to implement a distributed RDF repository. It is built on top of MAAN [30]. Data indexing and query processing mechanisms work as follows:
Data indexing. The data indexing model takes advantage of the structure of RDF triple as each element is used as a DHT key at the MAAN level. 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 the sake of clarity, let us consider the storage of the following three triples in the RDFPeers infrastructure: `<info:rdfpeers><dec:creator><info:mincai>`; `<info:mincai><foaf:name>"Min Cai"`; `<info:mincai> <foaf:age> "28" ^^<xmls:integer>`[10]. Figure 6 shows the storage process of these triples into a RDFPeers network with eight present peers in a 4-bit identifier space. It also depicts the finger tables of nodes N_6, N_{14}.

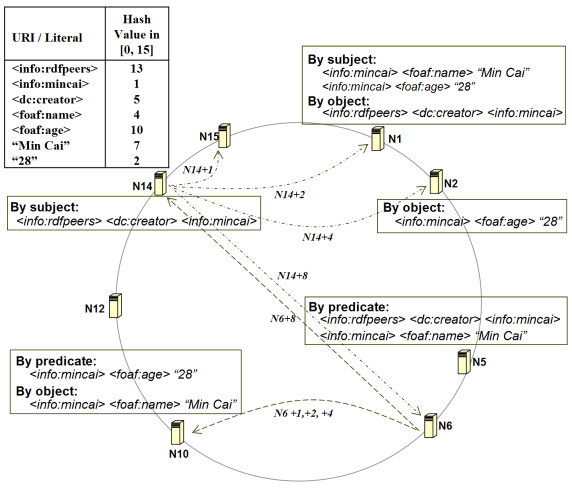

Figure 6: Example of RDF data storage in RDFPeers (from[31]).

Query processing. RDFPeers supports three kinds of queries.

- **Atomic triple pattern query** In this kind of query, the subject, predicate, or 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 $hash(s)$.

[10]Literals are either plain or typed. In the latter case, the datatype URI is appended to the literal.

All atomic queries take $O(log(N))$ routing hops to be resolved except queries in the form of $(?s, ?p, ?o)$ which require $O(N)$ 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 $(s, p, ?o), ?o \in \cup_{i=1}^{n}[l_i, u_i]$, such as l_i and 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 is forwarded from one node to the other, until it is received by the peer responsible for the upper bound of the last range. Disjunctive exact queries such as $(s, p, ?o), o? \in \{v_1, v_2\}$ 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, the 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 a such parameter. The pattern selectivity is also considered in [17] 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.

3.2.3 Atlas

A similar approach to RDFPeers was presented in Atlas project [74] which is a P2P system for RDF-S data storage and retrieval. It is built on top of Bamboo [102] and uses the RDFPeers for one-time query algorithm proposed in [31] and detailed in Section 3.2.2. Atlas uses RQL [67] as RDF query language. The added value of Atlas is that it combines one-time queries and *continuous queries* which will be discussed in Section 4.

3.2.4 Dynamic Semantic Space

In [55], Gu *et al.* propose a Dynamic Semantic Space (DSS), a schema-based P2P overlay network, where RDF statements are stored based on their semantics. Peers are organized in ring structure enabling the mapping from 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 overlay structure. Each cluster is identified by a *clusterID* given by a *k-bit* binary string such as $k = m + n$. While the m first bits identify the semantic cluster, denoted SC, the n bits represent the identifier of a cluster c which belongs to SC. Therefore, the semantic space infrastructure can have a maximum of 2^m semantic clusters and 2^n clusters per SC. For instance, if $m = 2$ and $n = 4$, there is at most $2^2 = 4$ semantic clusters and $2^4 = 16$ clusters per SC. As an example, Figure 7 depicts the architecture of a dynamic semantic space where 4 semantic clusters, SC_1, SC_2, SC_3, 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 c_2 and c_{16} are the two adjacent clusters of cluster c_1 where the peer p_0 belongs to, p_0 maintains references to peers p_2, p_3 such as $p_3 \in c_2$ ($c_2 \in SC_1$) and $p_2 \in c_{16}$ ($c_{16} \in SC_4$). Moreover, p_0 maintains pointers to other semantic clusters (i.e., SC_1 and SC_2).

Data indexing. The authors propose a two-tiers 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 suitable one, i.e., the most semantically related cluster.

Query processing. The query processing mechanism is performed in two phases. First, when the query, expressed in RDQL query language, is received, the peer pre-processes the query to get the information about the semantic clus-

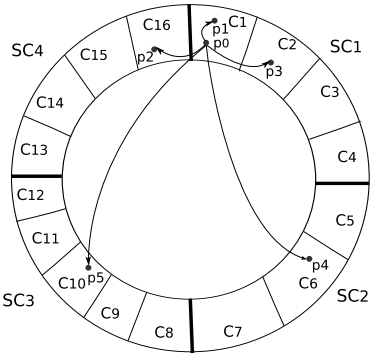

Figure 7: Architecture of Dynamic Semantic Space

ter, *SC* associated with it. When the query reaches the target semantic cluster *SC*, it will be flooded to all peers inside the semantic cluster. Peers that receive the query will perform a local search, and return results if available.

DSS provides a load balancing mechanism by controlling the number of peers per cluster and thus the global load. Therefore, a splitting and merging process can occur if the number of nodes per cluster reaches a given threshold value.

3.2.5 RDFCube

Monato *et al.* propose RDFCube [91], an indexing scheme of RDFPeers (Section 3.2.2) 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.

3.2.6 GridVine

GridVine [9, 40] is a distributed data management infrastructure. It is based on P-Grid DHT [8] at the overlay layer and maintains a *Semantic Mediation Layer* on top of it. 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 and structured data including schemas and schema mappings.

Data indexing. As in RDFPeers [31], GridVine indexes triples three times at the P-Grid layer based on their subjects, objects and predicates, i.e., the *insert (t)* operation results in *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. It supports the sharing of schemas. Each schema is associated with an unique key which is the concatenation of the logical address of the peer posting the schema, say $\pi(p)$, with the schema name, *Schema_Name*. Therefore, a schema indexing operation may look like *Update($\pi(p)$: 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..

Query processing. Lookup operations are performed by hashing the constant term(s) of the triple pattern. Once the key space is discovered, the query will be forwarded to peers responsible for that key space. As in P-Grid, the search complexity of atomic triples pattern is $O(log(\Pi))$ such as Π is the entire key partition. 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, whereas GridVine takes into consideration the schema heterogeneity which is not addressed in RDFPeers.

3.2.7 UniStore

Karnstedt *et al.* present their vision of a universal data storage at the Internet-scale for triples' repository through the UniStore project [65]. Their solution is based on P-Grid [8], on top of which they build a triple storage layer that enables the storage of data as well as metadata.

Data indexing. To face the challenge of data organization, authors chose to adopt a universal relation model allowing schema-independent query formulation. In order to speed up query processing and take advantage of the underlying features of the DHT for fast lookups, all attributes are indexed. When dealing with relational data, each tuple (OID, v_1, ..., v_n) of a given relation schema $R(A_1, \ldots, A_n)$ is stored in the form of n triples: (OID, A_1, v_1),...,(OID, A_n, v_n) where OID is a unique key, e.g., a URI, A_i an attribute name (that may contain a namespace prefix ns in order to avoid conflicts) and v_i its corresponding value. The OID field is not meant to be a unique and homogeneous identifier, instead, it is used to group the triples for a logical tuple. Each triple is indexed on the OID, the concatenation of A_i and v_i ($A_i \# v_i$) and finally on the value v_i.

Query processing. UniStore allows the use of a query language called Verti-

cal 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` and `WHERE` blocks, `FILTER`, `ORDER BY` 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. Operators can be applied to all levels of data (instance, schema and metadata). Each of these logical operators have a "physical"[11] implementation which solely relies on the functionalities provided by the overlay. 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.

Authors extended the UniStore project in [66] by proposing a distributed infrastructure for semantic data storage, querying and integration. In addition to the data indexing and querying mechanisms provided by the P-Grid overlay, the proposed infrastructure deals with the semantic clustering of triples and manage the mapping and relations between them. In the query processing phase, each query is first translated to a query plan and routed to the responsible peers. Once the target peer receives the query plan, it processes parts of it locally, inserts the (partial) results and forwards them to its appropriate neighbors, that is, nodes potentially responsible for the resolution of the next sub-queries. This process is repeated until the final results are received by the query initiator.

3.2.8 YARS

Yet Another RDF Store (YARS) proposed in [57] is a distributed RDF store that provides a set of facilities for RDF data management. The RDF data model introduced in this work extends the standard RDF data model (subject, predicate, object) 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 others scenario, RDF triples sharing the same semantics, can have the same context. Therefore, in YARS, each RDF triple, denoted by *t=(s,p,o)*, is associated a context, labeled *c*. The RDF triple combined with the context forms what is called *quad*. YARS uses the Notation3 (N3) [116] as a query language, combined with YARS query language which explicitly deals with the context concept. As the aim behind this indexing scheme is to efficiently support the evaluation of *Select-Project-Join* (SPJ) queries while minimizing the search cost, YARS takes advantage of *B+*- trees [38] for indexing as they support an efficient processing of range queries.

Data indexing. The proposed index structure is based on *lexicon* and *quad* indexes. The lexicon indexes operate on the string representation of RDF nodes

[11] Physical operators only rely on functionality provided by the overlay system (key lookup, multicasts,...). They differ in the kind of used indexes, applied routing strategy, parallelism, etc.

(i.e., each element of the RDF triple) and allow the retrieval of their object identifiers *(OIDs)*.

- **Lexicon indexes** consist of *nodeoid* , *oidnode* and the *keyword* indexes. While the *nodeoid* and the *oidnode* are used to map OIDs to node values and vice versa, the *keyword* indexes keep an inverted index to string literals in order to speed up full-text searches operations. More precisely, literals values are tokenized into words and each word is considered a key in the index. For each word, the number of occurrences and the OID to which it belongs to is maintained.

- **Quad indexes** allow the management of more complex queries. As each element of the quad *(s,p,o,c)* can be either specified or a variable, there is $2^4 = 16$ possible access patterns. Since the storage of 16 indexes for the same quad is quite expensive in terms of storage space, an optimized solution was proposed allowing the coverage of all access patterns using only 6 indexes rather than 16. Thereby, a single index can then be used to cover more than one access pattern. For instance, the *poc* index is used to process not only *(?,p,?,?)* but also *(?,p,o,?)* and *(?,p,o,c)*.

Query processing. Query resolution exploits both lexicon and quad indexes to support several queries types such as atomic queries, range queries, conjunctive queries. In particular, managing conjunctive queries requires a reordering of quads in order to start with the quads that yield the smallest result set and then combine it with the other results. An estimation of the result set of an access pattern can be based on the number of variables specified in it.

3.2.9 PAGE

Put And Get Everywhere (PAGE) presented in [44] is a RDF distributed repository based on Bamboo DHT [102] and implements YARS data indexing model discussed in Section 3.2.8. Therefore, as in YARS, PAGE manages quads rather than triples. Each quad is indexed six times and is 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 index used (i.e., *{spoc,cp,ocs,poc,csp,os}*) to build the quad ID. For the sake of clarity, 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 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 during the query routing. Take back the example discussed above and consider the access pattern *q=(x,y,?,?)* which looks, for the given subject x and the predicate y, for all object values in any context. This access pattern requires the

index ($idx(spoc) = 1$) to be resolved. For unknown parts (pointed out by ?) a sequence of 0 is assigned. The query will then be converted to "$11F00/3$". The mask value is 3 and means that the three first digits of the identifier are fixed. Thereby, only the last two elements (object and context) will be considered during the query processing.

3.2.10 Battré et al.

In [18], Battré *et al.* propose a data management strategy for DHT-based RDF store. As many others [31, 9, 91], the proposed approach indexes the RDF triple by hashing its subject, predicate and object. It takes into consideration RDF-S reasoning on top of DHTs by applying RDF-S reasoning rules.

Data indexing. The main characteristic of this approach compared to other RDF-based structured P2P approaches is that nodes host different RDF repositories, making a distinction between local and incoming knowledge. Each RDF repository managed by a node serves 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 hash space.
- **Received triples repository** stores the incoming triples sent by other nodes
- **Replica repository** ensures *triple availability* under high peer churn. The node with an ID numerically closest to the hash value of a triple becomes root node of the replica set. This node is responsible for sending all triples in its received database to the replica nodes[12].
- **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 specially for uneven key distribution. For instance, the DHT may store many triples with the same predicate *p="rfd:type"*. As 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 [58], one of the authors proposes a RDF query algorithm with optimizations based on a look-ahead technique and on Bloom filters [24]. Knowledge (i.e., RDF data) and queries are respectively

[12] In pastry, for example, the replica nodes are defined by the leaf set.

represented as *model* (T_M) and *query* (T_Q) 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. The first step works like this: 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 candidates where there is a value not matching 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 other optimization, using Bloom filters, goes like this: consider candidates for a triple (x, v_2, v_3), where x is a fixed value and v_2 and v_3 are variables. When retrieving the candidates by looking up using the fixed value x, i.e., executing $lookup(x)$, 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 v_2 and v_3. However, those sets may be large, that is why authors use Bloom filters to reduce the representation of the sets. 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.

3.2.11 Query Chain and Spread by Value algorithms

In [81], Liarou *et al.* propose two query processing algorithms to evaluate conjunctive queries over structured overlays: the Query Chain (QC) and Spread by Value (SBV) algorithms.

QC - Data indexing. As s in RDFPeers [31], in QC algorithm, a RDF triple is indexed to three nodes. More precisely, for a node p 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 nodes n_1, n_2 and n_3 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 formally, the query initiator, denoted by n, issues a query q composed of q_1, q_2,..., q_i patterns and forms a *query chain* by sending each triple pattern to a (possibly) different node, based on the hash value of constant part of each pattern. For each of the identified nodes, the message $QEval(q, i, R, IP(x))$ 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 (i.e., the node that poses the query) 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, authors adopt the default order for which the triple patterns appear in the query.

SBV - Data indexing. In the SBV algorithm, each triple $t = (s, p, o)$ 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 "+" operator denotes the concatenation of string values. By triple replication, the algorithm aims to achieve a better query load distribution at the expense of more storage space.

SBV - Query processing. SVB extends the QC algorithm in the sense that a query is processed by a 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 $q_1 \wedge \ldots \wedge q_k$ sends q to a node n_1 that is able to evaluate the first triple pattern 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 n_1 will use the values found locally that matches q_1, to bind the variables of $q_2 \wedge \ldots \wedge q_k$ that are in common with 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 [16] 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 [81], where intermediate results for a conjunctive query resolution are usually send through the network to nodes that store the appropriate data, Battré in [16], combines the fetching approach with the intermediate results forwarding approach. Such decision is taken at runtime and is based on the estimated traffic for each of both data integration techniques.

3.3 Complementary Techniques for Search Improvements

As the scalability of RDF-based P2P networks mainly depends on the way that queries are propagated and how much unnecessary data is transferred among peers in the network, this section discusses additional techniques that attempt to improve the RDF data retrieval in P2P systems such as caching, parallel query processing, etc.

P2P system	Network architecture	Data indexing	Queries: language, type and evaluation
Bibster [61]	Unstructured	• Semantic extraction of bibliographic metadata • Transform semantic data to ontologies through BibToOnto component	• SeRQL (Sesame RDF Query Language) • Ontology and peer's expertise-based forwarding • Use similarity function to measure the semantic similarity between the query content and the peer's expertise model
S-RDF [125]	Unstructured	• Triples are stored in RDF files • Triples are grouped by subjects	• Any query language is supported • Random walk and highest potentiality forwarding mechanisms
Edutella [97]	HyperCuP super-peer topology [109]	• Semantic clustering at the Super-Peer level • Maintain Super-Peer-Peer and Super-Super-Peer indices	• RDF-QEL (RDF Query Exchange Language) • Lookup at the super-peer level based on SP-SP indices • Lookup inside the cluster based on SP-P indices
RDFPeers [31]	MAAN-based [30]	• Three-times RDF triple indexing: $hash(s)$, $hash(p)$, $hash(o)$ via locality preserving $hash$ function	• Intuitions to transform RDQL queries into native RDFPeers queries • Support atomic, conjunctive, disjunctive queries with the *same* subject • Recursive multi-predicate query resolution
Atlas [74]	Bamboo-based [102]	• RDFPeers indexing scheme	• RQL (RDF Query Language) • RDFPeers query processing algorithm
DSS [55]	Chord based [114]	• Semantic clustering approach • RDF mapping to the most suitable semantic cluster	• RDQL (RDF-Data Query Language) • Routing inside the most suitable semantic cluster • Flooding inside the cluster
RDFCube [91]	CAN-based [101]	• Indexing scheme of RDFPeers • Store bit information regarding the existence of RDF triples	• Query mapping into a cell of RDFCube through hashing • Speed up "join queries" by performing `AND` operation on bit information
GridVine [9]	P-Grid-based [8]	• Semantic mediation layer overlay on top of P-Grid • P-Grid-based data indexing model except that RDF triples are indexed three times • Index schemas at the semantic mediation layer	• OWL (Web Ontology Language) for schema mapping • Support of atomic, conjunctive, disjunctive queries • Iterative query resolution
Unistore [65, 66]	P-Grid-based [8]	• Schema-independent query formulation: each tuple (OID, v_1, ..., v_n) of a given relation schema R(A_1, ..., A_n) is stored as n triples:(OID, A_1, v_1),..., (OID, A_n, v_n)	• VQL (Vertical Query Language) • VQL allowing DB-like and IR-like (similarity, ranking...) queries • Adaptive cost-aware query processing approach • Semantic layer on top of P-Grid
YARS [57]	B- tree [38]	• Manage *quad* (s,p,o,c) rather than RDF triple (s,p,o) • Maintain lexicon indexes operate on the string representation of RDF nodes • Maintain quad indexes based on the quad access pattern • Store *quad* in six indexes	• Notation3 (N3) + YARS query language • Support of atomic, range, conjunctive queries • For conjunctive query, starting by quads that yield to the smallest result
PAGE [44]	Bamboo-based [102]	• YARS indexing scheme	• Bamboo based routing • Use *mask* and *hops* properties during the query resolution
Battré et al. [18, 58]	DHT-based	• RDFPeers-like indexing mechanism • Nodes host different RDF repositories (local/ received/ replica/ generated)	• Graph-based models (for knowledge and queries) • Two-phase refinement procedure for query resolution • A look-ahead and Bloom filters enhancement techniques.
QC and SBV [81]	DHT-based	• RDFPeers indexing scheme	• The query language is not clearly specified • Query Chain (QC): 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 • Spread by Value (SBV): construct multiple chains for the same query

Table 2: 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. In YARS scheme, c denotes the *context* of the application.

3.3.1 Caching

To improve the search performance in P2P-based RDF stores, Battré introduced in [16] a caching approach that strives to reuse intermediate results of previous queries allowing therefore a quick process of new ones. More formally, let us denote by $T(Q)$ the set of the triples patterns forming a conjunctive query $H(Q)$ and by x the node that maintains the final results of the query $H(Q)$. The peer x computes an identifier $id = hash(H(Q), T(Q))$ which determines a peer' position in the DHT space. The peer y identified by id will store a reference to the node x. In others words, the peer y is aware that the result set of the query $H(Q), T(Q)$ can be found in peer x without fetching the result from x. Assume that, later, a peer z issues the same query $H(Q), T(Q)$, the query resolution mechanism starts by checking whenever the query result was already referenced at the peer identified by $hash(H(Q), T(Q))$ (which is the peer y in this example). If the peer z finds the reference on the peer y, then it fetches the result of $H(Q)$ from peer x. Otherwise, the query will be evaluated as in [81]. To limit the number of cached entries, only results that have been requested more than a given threshold will be stored. As a side note, caching query results have been also considered in S-RDF [125] (see Section 3.1.2) as well as in QC [81] (see Section 3.2.11).

3.3.2 Parallel RDF Query Processing

In [84], Lohrmann *et al.* proposed an algorithm aiming at parallelizing RDF query processing on top of DHTs and thus reduce the estimated response time. Each peer runs a *master agent* process that accepts connections from query clients and participates in the query evaluation. The proposed algorithm runs into the query *planning* and *execution* phases. Assume that a query $Q = \{q_1, \ldots, q_n,\}$ is submitted by a requester where q_i is a triple pattern. In the planning phase, and for each triple pattern q_i, the master agent identifies a peer with a $dhtId_i$ responsible for q_i by applying a hash function on the constant part of q_i. As in [17], if there is more than one constant part in the q_i, subject is chosen over object over predicate. For each q_i the master agent sends a `SREQ` message for the peer identified by the $dhtId_i$ Once the `SREQ` message is received, a slave agent process is triggered and the triple pattern is evaluated. A *pattern relation* α_i is computed such as $\alpha_i = (H_i, B_i)$, where H_i the pattern's variables, and B_i presents all possible valuations of the variables as given by the pattern matches. After receiving all replies from slave agents, the master agent has to find a *parallel join order* for the pattern relations and find a slave agent for each join operation in order to minimize the estimated execution time. Therefore, it sends a query execution plan (`QEP`) to each slave agent. The `QEP` describes a set of operations (e.g., join two local relations, send a relation to another peer) that have to be performed by each slave agent during the execution phase. Note that the `QEP` is the output of a tree join optimization algorithm run by the master agent after receiving all replies from the slave agents and aims to determine the dependency between sub-queries of the query Q. Each slave agent executes the corresponding `QEP` plan and returns the results to the master agent which, in turn, sends them to the query client.

3.4 P2P and RDF: Discussions and Challenges

The combination of P2P communication model with RDF data has become a very active research area aiming at sharing and processing a huge amount of data. In addition to the basic challenges related to such large scale infrastructure (network partition, network maintenance, etc.), enabling complex querying of RDF data on top of such infrastructure requires advanced techniques for data indexing and query processing algorithms. We will now summarize some challenges surrounding this combination and possible improvements with respect to the topology, data indexing and query processing.

Network topology. As we have already seen, a lot of work has been carried out towards a scalable distributed infrastructure of RDF data management. As one may noticed, most of the presented approaches choose the structured P2P networks as infrastructure basis(e.g., [9, 18, 31, 55, 57, 66, 74, 81, 91, 98, 44]). Such choice comes as a side effect of two main reasons. First, as it is already argued in [85], structured peer-to-peer overlay networks provide a useful substrate for building distributed applications due to the search guarantees that can be offered as long as the data is available in the network. Second, the structured nature of the RDF data model influences the choice of topology as it can be noticed through the data indexing model of the most of the P2P approaches. In this respect given the strong relationship between the network topology and the data indexing mechanism, the network model has to be carefully selected as it necessarily affect the performance of the data indexing method and undoubtedly the data lookup and query processing mechanisms.

Data indexing. Several approaches, e.g., RDFPeers [31], Atlas [74], YARS [57], RDFCube [91], Battré *et al.* in [18], while based on different overlay topologies, share almost the same data indexing model by *hashing* the RDF triple elements.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 and without needing to be collected from all data sources. However, individual nodes, responsible for overly popular triples (e.g., `rdf:type`, `dc:title`), can be easily overloaded resulting in poor performance. One possibility to address this issue is to use multiple hash functions to ensure a better load distribution as proposed in [101] and recently in [93]. The second category of RDF-based P2P approaches harness the semantic of RDF data either to build a "semantic" layer on top of the P2P overlay (e.g., [40, 66]) or to adopt a semantic clustering approach and organize the P2P layer as function of the semantic of the stored data (e.g., Edutella [98], DSS [55], S-RDF [125], Bibster [61]). Others have extended the basic RDF data model by adding the "context" concept as in YARS [57] and PAGE [44]. By taking into consideration data semantics in the data indexing phase, these approaches try to improve data lookup. However, this need 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., [18, 31, 40, 74, 81]). 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 [76], 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 [76], 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 [71]. 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 tradeoffs together and presented the CAP theorem [52] 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 needs is a fundamental step in building a successful distributed, scalable, highly reliable system.

Query processing. On 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., [81, 31]) whenever the query is partially resolved. However, we firmly 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 operaion between them conducts to a small set. *Caching* results [16, 81, 125] can alleviate this challenge, at least for similar queries, but can also affect the data consistency. Thereby, a tradeoff is made between the network resource usage on one side and the information staleness on the other side. Others approaches such as in [16] decide at runtime whether the current result set has to be fetched to the query's originator or continue to be forwarded to others neighbors[13]. As the query processing becomes more crucial especially when processing huge data set such as the Billion Triple Challenge 2009 (BTC2009) dataset[14], some works have been proposed to reduce the large data sets to the interesting subsets as in [54]. To do so, BitMat [12], based on Bit Matrix conjunctive query execution approach, is used to generated compressed RDF structure. Therefore, a dominant challenge related to the distributed query processing [73] 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 [21]. It is based on Jena ARQ[15], uses the concept of triple pattern selectivity[16] estimation[17]. It aims to find the query execution plan that minimizes the intermediate result set size of each triple pattern by join re-ordering strategies. Thereof, the

[13]The decision is taken based on the estimated traffic of each operation.
[14]http://vmlion25.deri.ie/.
[15]http://jena.sourceforge.net/ARQ
[16]Selectivity of triple patten T is the fraction of triples that satisfying this pattern.
[17]This is performed by collecting statistical information about the ontological resources.

smaller the selectivity the less intermediate results it is likely to produce and the earlier it should be executed in the query execution plan. For a similar purpose, RCQ-GA [59] uses genetic algorithms in order to perform an efficient evaluation of RDF chain queries.

All in all, the P2P communication model, combined with RDF data, constitutes a sound step for managing the tremendous growth of Web-scale RDF data set in a distributed environment. However, having high efficiency and scalability requirements in mind, RDF data management needs efficient algorithms for processing highly expressive queries.

4 Publish/Subscribe Communication Model

The publish/subscribe (pub/sub) communication model [49] gained a lot of interest and is considered as a powerful communication paradigm. The pub/sub model can be seen as an extension of the usual one-time query retrieval over P2P networks, allowing to efficiently receive notifications to long-standing subscriptions in a highly dynamic environment. In this section, we briefly highlight the main concepts underlining 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 delving into details of the subscription and advertisement processing algorithms, we will present some ground works that were done around this promising symbiosis.

4.1 Background

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 model and the filter model used.

- **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 [18]. An event notification service is the mediator 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 (such as IEEE Xplore) 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 [35], Sub-to-Sub [121]. 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 the RDF data management 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 [49, 83, 95] for deeper discussions on the basic concepts behind this communication model. In this part, the words "subscription" and "query" are interchangeable.

4.2 P2P-based Publish/Subscribe Systems for RDF Data Storage and Retrieval

4.2.1 Cai et al.

Cai *et al.* have proposed in [32] a content-based pub/sub layer extension atop their RDFPeers system [31] (see Section 3.2.2). In their system, each peer p

[18] In fully decentralized pub/sub systems, every node can perform the matching process.

maintains a list of local queries (i.e., subscriptions), along with the following information: triple patterns, the subscriber node identifier, a requested notification frequency and the requested subscription expiration date. Once a peer p stores (upon insertion) or removes a triple locally, it evaluates the matching subscription queries. This is also done after a triple is updated or when a collection of several matches for a given subscription is made. Afterwards, the peer notifies the subscriber by sending matched triples.

Subscription processing. Their subscription mechanisms support *atomic, disjunctive, range* and *conjunctive multi-predicate* queries[19,20]. The basic subscription mechanism for *disjunctive* and *range* queries is similar to that used for atomic queries, except that a subscription request is stored by all nodes that fall within the hashed identifiers of the minimum and maximum range value. The subscription scheme to manage *conjunctive multi-predicate* subscriptions is similar to the query chain algorithm discussed in Section 3.2.11. Initially, the subscription request S_r will be routed to the node p corresponding to the first triple pattern s in the subscription. Once p processes the first conjunct s, it removes s from S_r and stores it locally. In addition, p also stores the hash value of the next pattern in S. The node, receiving the subscription request (i.e., $S_r \setminus \{s\}$) sent by p stores the next pattern, and routes the remaining patterns towards the appropriate peer, and so on. The node that stores the last pattern in the subscription 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. Also, the authors propose an extension to support highly skewed subscriptions patterns, to avoid having the vast 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. 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 (structurally analog to the finger table, i.e. containing $log(N)$ nodes) 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 [30] (see Section 2.2.5), along with a repair protocol for the subscriptions and the data is provided (only for atomic queries). Replication can be tuned using a *Replica_Factor* parameter, thus, the subscription list will be replicated to the next replication factor nodes in the identifier space, i.e., it will send replicates to its immediate successors, following the replica factor parameter. The same goes for RDF triples. To manage situations where a user leaves while his subscriptions are still active, each subscription is characterized by a maximum duration parameter. This parameter controls the lifetime of the subscription. Therefore, if the duration d for a subscription s is expired, s will be removed from the subscription list.

[19]Not implemented at the time of the writing of their paper.

[20]Some combinations of conjunctive and disjunctive are not supported.

4.2.2 DSS

The Dynamic Semantic Space [55], investigated in Section 3.2.4, supports content-based pub/sub mechanism.

Subscription processing. Once a subscription request is generated it will be mapped to the corresponding semantic cluster (SC). From there, it will be forwarded to every node within the semantic cluster. Upon receipt of the subscription request, a peer p checks for a local matching of the subscription with its local RDF data. After a successful matching, it will send back 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 subscriber wants to unsubscribe to an event, an unsubscription request will be sent directly to the relevant producers.

4.2.3 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 [98] (see Section 3.2.1) and P2P-DIET [75] networks. Authors formalized the basic concepts of their content-based pub/sub system based on RDF using a *typed first-order* language $\mathcal{L}$. 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 shortly.

Advertisement indexing and processing. In this model, once a peer p 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 processing subscriptions. Advertisements contain one or more elements. These elements 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 that peers support. Each schema is uniquely identified by a URI. The schema indexing can be used, for example, to constrict the forwarding of the subscriptions only to the peers which support 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 final type of indexing is useful when a peer might choose to use only certain 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 p are inserted into the poset of its super-peer sp. As in the SIENA system [34], the poset consists of a set of subscriptions. The subscriptions' poset is actually a hierarchical structure of subscriptions which captures the notion of subscription subsumption, that is, when the result of a subscription q_1 is a subset of the result subscription q_2, we say that q_2 subsumes q_1. This concept is mainly used to avoid forwarding certain subscriptions which were subsumed by previously forwarded subscriptions, thus reducing network traffic. For each subscription, an associated *forwarder* list holds the subset of neighbors to which a subscription s has been forwarded and a *subscriber* list maintaining the subscriptions' originators. 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 Adv_1 sent by sp_2 to sp_1. If sp_1 has already received a subscription S (by some other peer) and there is a match with Adv_1, then sp_1 informs sp_2 not to forward any advertisement of this kind anymore, because the subscription responsible for S subsumes Adv_1. When a new notification n arrives at a super-peer sp, sp checks for the subscriptions satisfying n 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 p associated to a super-peer sp leaves the network, sp puts the subscriptions of p in a buffer for a period of time in order to receive notifications sent to p. Once p reconnects to sp, these notifications will then be forwarded to it. 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. This is actually pretty helpful especially knowing that after a leave (or a crash), a node may not have the same IP address, and thus might fail to recover its waiting subscriptions results. Their system guarantees that this will not happen.

4.2.4 Atlas

The Atlas infrastructure [74] in addition to *one-time* querying (see Section 3.2.3) also supports a content-based pub/sub paradigm, using algorithms from [81] detailed in Section 3.2.11. The system also supports the RDF Update Language (RUL) for RDF metadata insertion, deletion and update.

Subscription processing. After submitting an atomic subscription by a query originator p and being indexed in the network, the peer p waits for triples that satisfy it. 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 satisfy a query only partially by satisfying one of its sub-query. 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 of a query are satisfied.

4.2.5 Single and Multiple Query Chain Algorithms

Liarou *et al.* have proposed in [80] a content-based pub/sub system, which mainly focuses on conjunctive multi-predicate queries, and 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 3.2.11.

Subscription processing. In their pub/sub system, each triple t is characterized by a *published time* parameter, labeled $pubT(t)$. Moreover, each subscription S such as $S = s_1 \wedge s_2 \wedge ... \wedge s_n$ is identified by a $key(S)$ and has a timestamp indicating the *subscription time*, denoted by $subscrT(S)$. As a result, s_i is also characterized by a subscription time such as $subscrT(s_i) = subscrT(S)$. Therefore, a triple t can satisfy a subscription s of S *iff* $subscrT(s) < pubT(t)$. Subscriptions and notifications forwarding policies are similar to algorithms proposed in [81]. In the case of SQC, for each query, a single query chain is created at node r upon receipt of query S whereas MQC goes a step further. In MQC, first, a query S is indexed to a single node r according to one of S's sub-queries. Then, each time a triple arriving at r satisfies this sub-query, the subject is used to rewrite 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 n using the same predicate p will be answered when a new triple with the predicate p is inserted. In such scenario, only one matching operation has to be performed whenever such a triple is inserted.

4.2.6 Continuous Query Chain and Continuous Spread-By-Value Algorithms

In another work [79], Liarou *et al.* propose an extension of their two algorithms presented in [81] (QC and SBV). By introducing a *continuous* flavour of these algorithms, thus becoming CQC (*continuous query chain*) and CSBV (*continuous spread by value*), they modify them in order to be the core of a content-based pub/sub system.

CQC - Subscription processing. A node n that wants to subscribe with a conjunctive query $q = [q_1, \ldots, q_k]$ will do so by indexing each triple pattern q_j to a different node n_j. Thus, each of these nodes will be responsible for processing their part of the query q_j and will be a part of a *query chain* of q. Each node indexes the triple patterns as in QC, explained in Section 3.2.11.

When a node receives a newly indexed triple t, it will check any relevant indexed queries in its *query table (QT)*. If any match is found, there is a *valuation*[21] v over the variables of q_j such that $v(q_j) = t$. ***(i)***. If it is the first node in the chain, it forwards the valuation v (holding a partial answer to q) to the next node in the chain, n_i. When receiving this *intermediate result*, n_i will apply the received valuation and compute a new valuation w to the pattern q_i it holds, resulting in $q'_i = w(q_i)$. Then, it will try to find triples matching 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 $w' = w \cup v$. This new valuation is then

[21] Concept used to talk about values satisfying a query.

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 q is sent to the node that originally posed q. ***(ii)***. If the node is not the first in the query chain, it will store the new triple in its TT and search its IRT to see if 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 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. Node n can use the valuation v to rewrite $q = [q_1, \ldots, q_k]$ with fewer conjuncts $q' = [v(q_2), \ldots, v(q_n)]$ and decides *on the fly* the next node that will undertake the query processing. Because q' is conjunctive like q, its processing proceeds in a similar manner. Depending on the triples that trigger q_i, a node can have multiple next nodes for the processing of the query. Thus, the responsibility of evaluating the next triple pattern of q is distributed to multiple nodes compared to just one 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 (i.e. a concatenation) of all the constant parts will be used to index the query, following the triple indexing of SBV in Section 3.2.11.

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 (remember that the intermediate results depends on the reception of a new triple and its potential valuation).

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 (rate of published triples that trigger a triple pattern, etc.).

4.2.7 Ranger et al.

In [100], the authors introduce an information sharing platform based on Scribe [35], a topic-based[22] pub/sub system. 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 sub-

[22] Actually, it is not a "pure" topic-based ; in this work a topic is a query.

scribe to a topic (i.e., to a query) and to lookup within the group[23] of the topic. Frequent queries are cached effectively as they occur, remaining active as long as client 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 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 just by taking a look at it. This depends on whether objects are stored as a whole on each 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, vCard, rdf, 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, that is, subscribes to a topic, it first looks up in the group for a peer that already knows (or is interested to know) the result of the query. If one 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 8 shows the result pipelines established when a peer P1 performs a query.

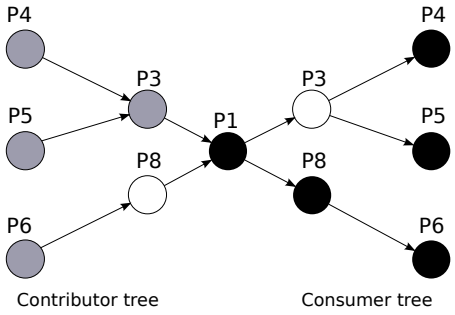

Figure 8: Contributor and consumer pipelines resulting from P1 performing a query. Gray nodes are contributors to query Q; white nodes are forwarders[25]; black nodes are consumers of the results integrated by P1. (figure taken and redrawn from [100])

Within a group, whenever a peer is processing an atomic query, it will broadcast the fact to others peers using the underlying group communication primitives offered by Scribe. 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

[23] A group is made of all the peers interested in the same query.

[25] Forwarders are nodes which help in the dissemination process within a tree even if they are not interested in the content of the events they propagate. This feature directly derives from Scribe.

for this is to avoid ignoring any potential source of results. The algorithm makes use of Pastry's built-in mechanisms 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.

4.3 Pub/sub and RDF: Discussions and Challenges

Pub/sub systems have been studied for some time now and are well established, but there subsists a strong need for Quality of Service (QoS) [88], especially when deployed on top of a *best-effort* infrastructure such as the Internet. We will discuss a non-exhaustive list of improvement opportunities, with an emphasis on QoS, which are applicable to our specific combination of interest, that is, the publish/subscribe paradigm coupled with the RDF data model.

Research on this combination is still ongoing and there is room for many improvements both at the subscription and notification processing levels.

The presented works have slightly different mechanisms for dealing with notifications and subscriptions. Most of them, being based on structured peer-to-peer overlay directly take advantage of the underlying indexing mechanisms. Triples are indexed and retrieved using cryptographic functions (except for [100]) and this, generally, for each RDF triple element and their combination. This multi-indexing technique induces a non negligible processing load, and has pros and cons as discussed in Section 3.4. Ranger *et al.* do not index data *a priori* but reuse the subscription management and group communication primitives used for notifications' propagation offered by Scribe [35]. By adding several caching mechanisms, in order to maintain the most popular queries and respective results fresh, they enhanced the data retrieval mechanism. The only work based on a unstructured overlay network [10] relies on super-peers to manage and process subscriptions as well as route notifications.

Improvements related to *subscription processing.* Several techniques do exist in order to improve the processing of subscriptions. Subscription *subsumption* [63] and *summarization* [117] are two optimization techniques whose goals are to reduce subscription dissemination traffic and enhanced 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. Core pub/sub systems such as SIENA [34], REBECA [94] and PADRES [78] implement *pair-wise* subscription cover checking to reduce redundancy in subscription forwarding. A more efficient approach, argued in [63], would be to exploit subscription subsumption relationship between a new subscription and a set of existing active ones. In the presented pub/sub systems, only one [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 [22, 23]. For some other type of systems, such as peer-to-peer streaming video, a more *relaxed* reliability would be more appropriate since it will not make a lot of sense to

Pub/Sub system	Direct related work(s)	Contribution details
Cai et al. [32]	extends RDF-Peers [31]	• Subscription to atomic, disjunctive, range and conjunctive multi-predicate queries is possible. • Some combinations of conjunctive and disjunctives queries are not supported in subscriptions. • A replication mechanism is provided for both subscriptions and data (for atomic queries), inherited from MAAN [30]. • Support for highly skewed subscription patterns is offered.
DSS [55]	influenced by Small-World [70]	• Subscriptions are mapped to a semantic cluster and forwarded to all nodes within the cluster to be locally processed.
Chirita et al. [10]	influenced by Edutella [97] and P2P-DIET [75]	• Super-peer-based system. • Formalism of their system using a *typed first-order* language is given. • Resources advertisement managed by the super-peers using local partially order sets (reducing network traffic). • Subscriptions are arranged in a hierarchical structure and take advantage of *subscription subsumption*. • Notification matching and offline notifications' management are done by super-peers. • Integrated authentication mechanisms.
ATLAS [74]	extends Bamboo [102]	• RDF Update Language (RUL) for meta-data insertion, deletion and update. • Subscriptions and matching mechanisms directly derive from QC and SBV [81].
SQC and MQC [80]	extends Idreos *et al.* [60] and Tryfonopoulos *et al.* [118]	• 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 (Section 3.2.11): – *Single Query Chain* (SQC) – *Multiple Query Chain*(MQC)
CQC and CSBV [79]	extends Liarou *et al.* [81, 80]	• 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. [100]	based on Scribe [35]	• 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.

Table 3: P2P-based Publish/Subscribe systems for RDF data storage and retrieval.

re-send lost frames for live feeds. But still, ensuring a correct dissemination of streams despite failures does require a particular form of resiliency. 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 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 [88], 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 most of the interesting challenges:

(i) Delivery semantics. Delivery semantics come into play at the last hop, prior to notifying the subscriber. Depending on the network reliability and the support for duplicate message, notifications can be delivered in a *best effort* way (and thus can be duplicated at the destination); they can follow an *at most once* semantic which guarantees that a subscriber will receive only one notification (even if the notification was duplicated in the medium); the *at least once* delivery semantic ensures that the subscriber will receive at least one notification of an event instance (but duplicate instances can be received); finally the *exactly once* delivery semantic guarantees the reception of a single instance of a notification (but duplicates can be cached at intermediate nodes to ensure reliability). All these delivery guarantees are implemented by a wide variety of broadcast algorithms which can be found in [56]. 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 peer-to-peer systems.

(ii) Reliability. Also, 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 [87]. Earlier works at the overlay layer such as RON [11] and TAROM [115] 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 (load, subscriber/publishers degree, churn rate, etc). Hermes, an event-based middleware based on Pastry [104], 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 authors in [89] pointed out.

(iii) Message ordering. In pub/sub systems, some applications, such as the ones cited above (financial systems, air traffic control, Facebook feeds) emphasize on the preservation of a *temporal* order between the reception of events. The most common message orderings are: random, FIFO/LIFO, priority, causal or total. 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. Detecting causal relationships in distributed computations is far from being a trivial task as argued in [111], perhaps because computer scientists misunderstand it as reported in a famous controversy [119]. Maybe this misinterpretation finds its roots in physics, where the very notion of causality was greatly criticized by prominent physicist Bertrand Russel in [105] and more recently by Price [99]. Borrill perfectly perceived this confusion, and from a

distributed systems perspective raises interesting questions worthy of new reflections in the light of modern relativistic physics theories [25].

Strong dissemination abstractions such as a *Reliable Causal Broadcast*, well known and well studied in the traditional 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 [56] and in [43]. Since the majority of the systems presented are built on top of structured overlay networks, one should take advantage of the present structure to disseminate events in a smarter way, as in Scribe [35] or using efficient dissemination algorithms atop structured p2p networks [47]. The problem with Scribe is that it only provides best effort guarantees, even if authors state that strengthening the algorithm to make it reliable should not be that difficult. One possible solution, that can be used for a large family of structured overlay networks, would be the use of a *pseudo-reliable* broadcast abstraction as presented in [51] which ensures the correct dissemination of messages provided the root of the tree which is constructed and used to broadcast the events does not crash.

The study of these algorithms is now being revisited by the Distributed Systems research community in the context of *dynamic distributed systems* [15, 14]. A dynamic distributed system is, according to the same authors, *a continuously running system in which an arbitrary large number of processes are part of the system during each interval of time and, at any time, any process can directly interact with only an arbitrary small part of the system.* This informal definition shed a light on the nature of such systems, which is a coupling of two distinct concepts that are *large-scale* and *dynamicity*. Pub/sub systems fall into the realm of this definition combining the two aforementioned concepts; not only publishers and subscribers come and go at will (churn), but data itself is dynamic (events are continuously fed into the system, sometimes rendering prior events obsolete).

Another key point in large-scale and highly dynamic system is its ability to manage complexity. In order to face an increasing complexity, realizing autonomy is crucial [68]. This well known vision from IBM took the form of "special" properties in complex systems, namely *self-* properties* [13]. Properties such as self-management [120] and self-stabilization [45] are two prominent properties in decentralized systems and researchers apply these two concepts to pub/sub systems [62], in the case of self-management and in [96] in the case of self-stabilization. We could easily envision the usage of such algorithmic concepts in the context of RDF-based pub/sub systems as well as studying other self-* properties in this context.

Finally a ultimate challenge would be to consider pub/sub systems in extremely large-scale settings (at *exascale*[26]) for which there is a strong need to design new algorithms and data structures [46]. Again, from a QoS point of view, much has to be done [33]. The successful deployment of RDF in pub/sub systems at the Internet-scale is somehow correlated with solving it at the exascale.

[26] http://www.exascale.org

5 Conclusion and Perspectives

5.1 Summary

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 [90]. Peer-to-Peer technologies addresses *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 [48] - represents an important step towards practical scalable systems.

This survey has presented various research efforts on RDF data management in P2P systems emphasizing on RDF data indexing, retrieval and integration. The motivation behind this survey was to attempt to answer the following question: "*how RDF data have to be indexed in a distributed P2P environment to be efficiently retrieved ?*". 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 goal remains the following: to build large scale distributed applications using the P2P technologies in order to come up with fully distributed and scalable infrastructure for RDF data storage. The idea of clustering semantically related data in (unstructured) P2P networks was at the heart of Crespo *et al.* 's work [39], coining the term *semantic overlay network*. They presented methods for grouping and classifying data using hierarchies and show how search can be improved when data is grouped. However, they did not mention any specific data model or took advantage of the lookup guarantees offered by structured overlay networks.
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 points found at the P2P level, they incorporate research issues of their own as shown in Section 4.2: subscription processing and notification propagation. A full taxonomy of the presented works, along with their relative relationships, is depicted in Figure 9.

5.2 Perspectives

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. Garces *et al.* in [77] introduced a two-tier hierarchical structure using Chord at the top level and another DHT at the lower level. This structure, mimicking *Autonomous Systems* (ASes) generally found on the Internet, is used to connect heterogeneous groups (at the lower level) together. This is done by having a subset of peers, within each group, acting as gateways at the top level. This hierarchical structure

inspired the work proposed by Baude *et al.* [19] to create *Semantic Spaces*. A semantic space groups together semantically related RDF triples by storing them in a three dimensional CAN overlay network. These spaces are referenced at the top level by Chord nodes. This structure has several benefits in terms of QoS management since a semantic space is administered by a single organizational group which can react faster in case of problems for instance.

Research on multi-dimensional data management on top of decentralized systems is also an interesting correlated research stream [108, 110, 123, 124]. This branch of research can be seen as a generalization or an abstraction of RDF data management on top of P2P systems. In this regard, we believe that readers, interested in a deeper understanding of more generalized data management in P2P systems, will greatly benefit from the works presented in this survey. The reason is that they provide a solid and concrete first approach towards a more complex and more abstract data management in large-scale P2P systems.

A multitude of open research problems and directions are laid out in Sections 3.4 and 4.3. QoS management in such systems is still in its nascent stages and represents a considerable research issue. In order to fully grasp all the issues related with dynamism in P2P and pub/sub systems, we need to have sound models to reason analytically on the very nature of churn, such as the ones presented in [72].

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.

Acknowledgement

The presented work is funded by the EU FP7 NESSI strategic Integrated Project SOA4All (`http://www.soa4all.eu`) and EU FP7 Specific Targeted Research Project PLAY (`http://www.play-project.eu`).

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/`.

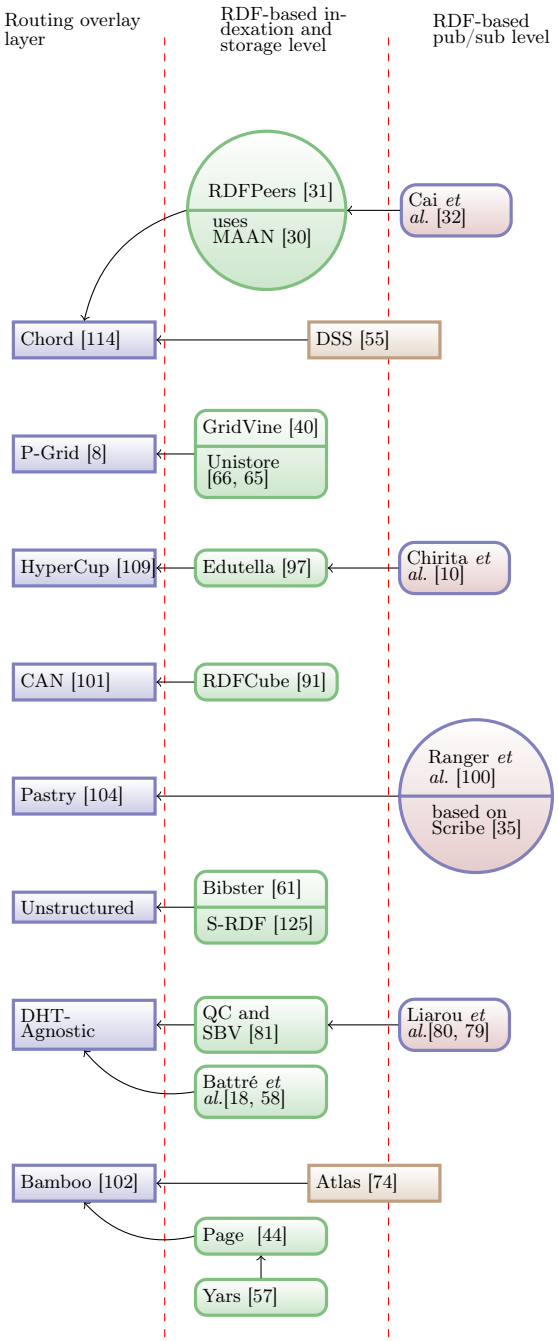

Figure 9: Taxonomy of the presented works and their relative connections to other works. The arrows carry an "extend/reuse" semantic. DSS and Atlas overlap two categories because they support *one-time* and *continuous* queries.

[8] K. Aberer, P. Cudré-Mauroux, A. Datta, Z. Despotovic, M. Hauswirth, M. Punceva, and R. Schmidt. P-Grid: a self-organizing structured P2P system. *ACM SIGMOD Record*, 32(3):33, 2003.

[9] Karl Aberer, Philippe Cudré-Mauroux, Manfred Hauswirth, and Tim Van Pelt. GridVine: Building Internet-Scale Semantic Overlay Networks. In *International Semantic Web Conference*, 2004.

[10] P. Alex, R. Chirita, S. Idreos, M. Koubarakis, and W. Nejdl. Designing Semantic Publish/subscribe Networks using Super-Peers. In *Semantic Web and Peer-To-Peer*, January 2004.

[11] David Andersen, Hari Balakrishnan, Frans Kaashoek, and Robert Morris. Resilient overlay networks. In *Proceedings of the 18th ACM symposium on Operating systems principles*, pages 131–145, Banff, Alberta, Canada, 2001. ACM.

[12] Medha Atre, Jagannathan Srinivasan, and James Hendler. BitMat: A Main-memory Bit Matrix of RDF Triples for Conjunctive Triple Pattern Queries. In *7th International Semantic Web Conference (ISWC)*, October 2008.

[13] Ozalp Babaoglu, Mark Jelasity, Alberto Montresor, Christof Fetzer, Stefano Leonardi, Aad van Moorsel, and Maarten van Steen. *Self-star Properties in Complex Information Systems*. Springer, 2005.

[14] R. Baldoni and A.A. Shvartsman. Theoretical aspects of dynamic distributed systems: report on the workshop, Elche, Spain, September 26, 2009. *Special Interest Group on Algorithms and Computation Theory (SIGACT) News*, 40(4):87–89, 2010.

[15] Roberto Baldoni, Marin Bertier, Michel Raynal, and Sara Tucci-Piergiovanni. Looking for a definition of dynamic distributed systems. In *Parallel Computing Technologies*, pages 1–14. Springer, 2007.

[16] Dominic Battre. Caching of intermediate results in DHT based RDF stores. *International Journal on Metadata Semantics and Ontologies*, 3(1):84–93, 2008.

[17] Dominic Battré. Query Planning in DHT Based RDF Stores. In *Proceedings of the 2008 IEEE International Conference on Signal Image Technology and Internet Based Systems (SITIS)*, pages 187–194, Washington, DC, USA, 2008. IEEE Computer Society.

[18] Dominic Battré, Felix Heine, André Höing, and Odej Kao. On Triple Dissemination, Forward-Chaining, and Load Balancing in DHT Based RDF Stores. In *DBISP2P*, pages 343–354, 2006.

[19] Françoise Baude, Imen Filali, Fabrice Huet, Virginie Legrand, Elton Mathias, Philippe Merle, Cristian Ruz, R. Krummenacher, E. Simperl, C. Hamerling, and J.-P. Lorré. ESB Federation for Large-Scale SOA. In *Proceedings of the ACM Symposium on Applied Computing (SAC)*, pages 2459–2466, 2010.

[20] T. Berners-Lee. Linked data. *W3C Design Issues*, 2006.

[21] Abraham Bernstein, Christoph Kiefer, and Markus Stocker. OptARQ: A SPARQL Optimization Approach based on Triple Pattern Selectivity Estimation. Technical report, University of Zurich, 2007.

[22] K.P. Birman. A Review of Experiences with Reliable Multicast. *Software: Practice and Experience*, 29(9):741–774, 1999.

[23] K.P. Birman, R.V. Renesse, and R. Van Renesse. *Reliable Distributed Computing with the Isis Toolkit*. IEEE Computer Society Press Los Alamitos, CA, USA, 1994.

[24] Burton H. Bloom. Space/Time Trade-offs in Hash Coding With Allowable Errors. *Communications of the ACM*, 13(7):422–426, 1970.

[25] Paul Borrill. Smart data and wicked problems. 2008.

[26] Djelloul Boukhelef and Hiroyuki Kitagawa. Multi-ring infrastructure for content addressable networks. In *Proceedings of the OTM 2008 Confederated International Conference*, pages 193–211, Berlin, Heidelberg, 2008. Springer-Verlag.

[27] Djelloul Boukhelef and Hiroyuki Kitagawa. Dynamic load balancing in rcan content addressable network. In *Proceedings of the 3rd International Conference on Ubiquitous Information Management and Communication (ICUIMC)*, pages 98–106, New York, NY, USA, 2009. ACM.

[28] Jeen Broekstra and Arjohn Kampman. SeRQL: An RDF Query and Transformation Language. August 2004.

[29] Jeen Broekstra, Arjohn Kampman, and Frank Van Harmelen. Sesame: An Architecture for Storing and Querying RDF Data and Schema Information. In *Semantics for the WWW*. MIT Press, 2001.

[30] Min Cai, Martin Frank, Jinbo Chen, and Pedro Szekely. MAAN: A multi-attribute Addressable Network for Grid Information Services. In *Journal of Grid Computing*, volume 2, 2003.

[31] Min Cai and Martin R. Frank. RDFPeers: a scalable distributed RDF Repository Based on a Structured Peer-to-Peer Network. In *WWW*, pages 650–657, 2004.

[32] Min Cai, Martin R. Frank, Baoshi Yan, and Robert M. MacGregor. A subscribable Peer-to-Peer RDF Repository for Distributed Metadata Management. *J. Web Sem.*, 2(2):109–130, 2004.

[33] Franck Cappello, Al Geist, Bill Gropp, Sanjay Kale, Bill Kramer, and Marc Snir. Toward exascale resilience. Technical Report TR-JLPC-09-01, INRIA, 2009.

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

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

[36] Miguel Castro, Manuel Costa, and Antony Rowstron. Peer-to-peer overlays: Structured, Unstructured, or Both. Technical Report MSR-TR-2004-73, Microsoft Research, 2004.

[37] Miguel Castro, Manuel Costa, and Antony Rowstron. Debunking Some Myths about Structured and Unstructured Overlays. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design and Implementation (NSDI)*, pages 85–98. USENIX Association, 2005.

[38] Douglas Comer. The ubiquitous b-tree. *ACM Computing Surveys*, 11:121–137, 1979.

[39] A. Crespo and H. Garcia-Molina. Semantic Overlay networks for P2P Systems. *Agents and Peer-to-Peer Computing*, pages 1–13, 2005.

[40] Philippe Cudré-Mauroux, Suchit Agarwal, and Karl Aberer. Gridvine: An infrastructure for peer information management. *IEEE Internet Computing*, 11:36–44, 2007.

[41] R.V. Guha Dan Brickley. RDF Vocabulary Description Language 1.0: RDF schema. http://www.w3.org/TR/rdf-schema/.

[42] Anwitaman Datta, Manfred Hauswirth, Renault John, Roman Schmidt, and Karl Aberer. Range queries in trie-structured overlays. In *P2P '05: Proceedings of the Fifth IEEE International Conference on Peer-to-Peer Computing*, pages 57–66, Washington, DC, USA, 2005.

[43] R. de Juan, H. Decker, E. Miedes, J. E. Armendariz, and F. D. Munoz. A Survey of Scalability Approaches for Reliable Causal Broadcasts. Technical Report ITI-SIDI-2009/010, 2009.

[44] E. Della Valle, A. Turati, and A. Ghioni. PAGE: A distributed infrastructure for fostering RDF-based interoperability. In *Distributed Applications and Interoperable Systems (DAIS)*, pages 347–353. Springer, 2006.

[45] E.W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11):643–644, 1974.

[46] Greg Eisenhauer, Matthew Wolf, Hasan Abbasi, and Karsten Schwan. Event-based systems: opportunities and challenges at exascale. In *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, pages 1–10, Nashville, Tennessee, 2009. ACM.

[47] Sameh El-Ansary, Luc Alima, Per Brand, and Seif Haridi. Efficient Broadcast in Structured P2P Networks. In *Peer-to-Peer Systems II*, pages 304–314. Springer, 2003.

[48] Sameh El-Ansary and Seif Haridi. An overview of structured overlay networks. In *Theoretical and Algorithmic Aspects of Sensor, Ad Hoc Wireless and Peer-to-Peer Networks*. CRC Press, 2005.

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

[50] Jun Gao and Peter Steenkiste. An adaptive protocol for efficient support of range queries in dht-based systems. In *Proceedings of the 12th IEEE International Conference on Network Protocols (ICNP)*, pages 239–250, Washington, DC, USA, 2004. IEEE Computer Society.

[51] Ali Ghodsi. *Distributed k-ary System: Algorithms for Distributed Hash Tables*. Thesis, KTH - Royal Institute of Technology, 2006.

[52] Seth Gilbert and Nancy Lynch. Brewer's Conjecture and the Feasibility of Consistent Available Partition-Tolerant Web Services. In *ACM SIGACT News*, page 2002, 2002.

[53] Li Gong. JXTA: A Network Programming Environment. *IEEE Internet Computing*, 5(3), 2001.

[54] Medha Atre Gregory Todd Williams, Jesse Weaver and James A. Hendler. Scalable Reduction of Large Datasets to Interesting Subsets. In *8th International Semantic Web Conference*, 2009.

[55] Tao Gu, Hung Keng Pung, and Daqing Zhang. Information Retrieval in Schema-based P2P Systems Using One-dimensional Semantic Space. *Computer Networks*, 51(16):4543–4560, 2007.

[56] R. Guerraoui and L. Rodrigues. *Introduction to reliable distributed programming*. Springer-Verlag New York Inc, 2006.

[57] Andreas Harth and Stefan Decker. Optimized Index Structures for Querying RDF from the Web. In *Proceedings of the Third Latin American Web Congress (LA-WEB)*, page 71, Washington, DC, USA, 2005. IEEE Computer Society.

[58] Felix Heine. Scalable p2p based RDF querying. In *Proceedings of the 1st international conference on Scalable information systems (InfoScale)*, page 17, New York, NY, USA, 2006. ACM.

[59] A. Hogenboom, V. Milea, F. Frasincar, and U. Kaymak. RCQ-GA: RDF Chain Query Optimization Using Genetic Algorithms. In *Proceedings of the 10th International Conference on E-Commerce and Web Technologies*, pages 181–192. Springer-Verlag, 2009.

[60] S. Idreos, C. Tryfonopoulos, and M. Koubarakis. Distributed Evaluation of Continuous Equi-join Queries over Large Structured Overlay Networks. In *Proceedings of the 22nd International Conference on Data Engineering (ICDE)*, pages 43–43, 2006.

[61] J. Broekstra and Marc 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.

[62] M.A. Jaeger. *Self-Managing Publish/Subscribe Systems*. PhD thesis, Technischen Universitat Berlin, 2007.

[63] Hojjat Jafarpour, Bijit Hore, Sharad Mehrotra, and Nalini Venkatasubramanian. Subscription Subsumption Evaluation for Content-based Publish/Subscribe Systems. In *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*, pages 62–81. Springer-Verlag New York, Inc., 2008.

[64] David Karger, Eric Lehman, Tom Leighton, Mathhew Levine, Daniel Lewin, and Rina Panigrahy. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *In ACM Symposium on Theory of Computing*, pages 654–663, 1997.

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

[66] Marcel Karnstedt, Kai-Uwe Sattler, Manfred Hauswirth, and Roman Schmidt. A DHT-based Infrastructure for Ad-hoc Integration and Querying of Semantic Data. In *Proceedings of the 2008 international symposium on Database engineering and applications (IDEAS)*, pages 19–28, New York, NY, USA, 2008. ACM.

[67] Gregory Karvounarakis, Sofia Alexaki, Vassilis Christophides, Dimitris Plexousakis, Forth Vassilika Vouton, and Michel Scholl. RQL: A Declarative Query Language for RDF. In *Proceedings of the 11th international conference on World Wide Web (WWW)*, pages 592–603. ACM Press, 2002.

[68] J.O. Kephart and D.M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[69] Raddad Al King, Abdelkader Hameurlain, and Franck Morvan. Query Routing and Processing in Peer-to-Peer Data Sharing Systems. *International Journal of Database Management Systems*, pages 116–139, May 2010.

[70] J. Kleinberg. The small-world phenomenon: an algorithm perspective. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 163–170. ACM New York, NY, USA, 2000.

[71] P. Knežević, A. Wombacher, and T. Risse. DHT-Based Self-adapting Replication Protocol for Achieving High Data Availability. *Advanced Internet Based Systems and Applications*, pages 201–210, 2009.

[72] Steven Y. Ko, Imranul Hoque, and Indranil Gupta. Using tractable and realistic churn models to analyze quiescence behavior of distributed protocols. In *IEEE Symposium on Reliable Distributed Systems*, volume 0, pages 259–268, Los Alamitos, CA, USA, 2008. IEEE Computer Society.

[73] Donald Kossmann. The State of The Art in Distributed Query Processing. *ACM Computing Surveys*, 32(4):422–469, 2000.

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

[75] M. Koubarakis, C. Tryfonopoulos, S. Idreos, and Y. Drougas. Selective information dissemination in P2P networks: problems and solutions. *ACM SIGMOD Record*, 32(3):71–76, 2003.

[76] Salma Ktari, Mathieu Zoubert, Artur Hecker, and Houda Labiod. Performance Evaluation of Replication Strategies in DHTs Under Churn. In *Proceedings of the 6th International Conference on Mobile and Ubiquitous Multimedia (MUM)*, pages 90–97, New York, NY, USA, 2007. ACM.

[77] PA Felber L Garces-Erice, EW Biersack and G Urvoy-Keller. Hierarchical Peer-to-peer Systems. In *Proceedings of ACM/IFIP International Conference on Parallel and Distributed Computing (Euro-Par)*, pages 643–657, 2003.

[78] G. Li, S. Hou, and H.A. Jacobsen. 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*, pages 447–457, 2005.

[79] E. Liarou, S. Idreos, and M. Koubarakis. Continuous RDF Query Processing over DHTs. *Lecture Notes in Computer Science*, 4825:324–339, 2008.

[80] Erietta Liarou, Stratos Idreos, and Manolis Koubarakis. Publish/Subscribe with RDF Data over Large Structured Overlay Networks. In *DBISP2P*, pages 135–146, 2005.

[81] Erietta Liarou, Stratos Idreos, and Manolis Koubarakis. Evaluating Conjunctive Triple Pattern Queries over Large Structured Overlay Networks. In *Proceedings of 5th International Semantic Web Conference (ISWC)*, 2006.

[82] Meng-Jang Lin and Keith Marzullo. Directional gossip: Gossip in a wide area network. In *EDCC*, pages 364–379, 1999.

[83] Ying Liu and Beth Plale. Survey of Publish Subscribe Event Systems. Technical report, Indiana University, 2003.

[84] Björn Lohrmann, Dominic Battré, and Odej Kao. Towards parallel processing of rdf queries in dhts. In *Globe*, pages 36–47, 2009.

[85] Keong Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim. A Survey and Comparison of Peer-to-Peer Overlay Network Schemes. *IEEE Communications Surveys and Tutorials*, pages 72–93, 2005.

[86] Qin Lv, Pei Cao, Edith Cohen, Kai Li, and Scott Shenker. Search and replication in unstructured peer-to-peer networks. In *ICS '02: Proceedings of the 16th international conference on Supercomputing*, pages 84–95. ACM, 2002.

[87] Shruti P. Mahambre and Umesh Bellur. An Adaptive Approach for Ensuring Reliability in Event Based Middleware. In *Proceedings of the second international conference on Distributed event-based systems*, pages 157–168. ACM, 2008.

[88] Shruti P. Mahambre, Madhu Kumar S.D, and Umesh Bellur. A Taxonomy of QoS-Aware, Adaptive Event-Dissemination Middleware. *IEEE Internet Computing*, 11(4):35–44, 2007.

[89] SP. Mahambre and U. Bellur. Reliable Routing of Event Notifications over P2P Overlay Routing Substrate in Event Based Middleware. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1–8, 2007.

[90] David Maier, Jeffrey D. Ullman, and Moshe Y. Vardi. On the Foundations of the Universal Relation Model. *ACM Transactions on Database Systems (TODS)*, 9(2):283–308, 1984.

[91] Akiyoshi Matono, Said Pahlevi, and Isao Kojima. RDFCube: A P2P-Based Three-Dimensional Index for Structural Joins on Distributed Triple Stores. *Databases, Information Systems, and Peer-to-Peer Computing*, pages 323–330, 2007.

[92] Elena Meshkova, Janne Riihijärvi, Marina Petrova, and Petri Mähönen. A Survey on Resource Discovery Mechanisms, Peer-to-Peer and Service Discovery Frameworks. *Comput. Netw.*, 52(11):2097–2128, 2008.

[93] Yuqi Mu, Cuibo Yu, Tao Ma, Chunhong Zhang, Wei Zheng, and Xiaohua Zhang. 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)*, pages 5364–5367, Piscataway, NJ, USA, 2009. IEEE Press.

[94] Gero Muhl. Large-Scale Content-based Publish/Subscribe Systems. *Ph.D Dissertation, Darmstadt University of Technology, Germany*, 2002.

[95] Gero Mühl, Ludger Fiege, and Peter Pietzuch. *Distributed Event-Based Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[96] Gero MÃijhl, Michael A. Jaeger, Klaus Herrmann, Torben Weis, Andreas Ulbrich, and Ludger Fiege. Self-stabilizing publish/subscribe systems: Algorithms and evaluation. In *17th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM)*, pages 233–238. Springer, 2005.

[97] Wolfgang Nejdl, Boris Wolf, Changtao Qu, Stefan Decker, Michael Sintek, Ambjörn Naeve, Mikael Nilsson, Matthias Palmer, and Tore Risch. EDUTELLA: A P2P Networking Infrastructure Based on RDF. In *Proceedings of the 11 International World Wide Web Conference (WWW)*, May 2002.

[98] Wolfgang Nejdl, Martin Wolpers, Wolf Siberski, Christoph Schmitz, Mario Schlosser, Ingo Brunkhorst, and Alexander Löser. 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*, pages 536–543. ACM, 2003.

[99] H. Price and R. Corry. *Causation, physics, and the constitution of reality: Russell's republic revisited*. Oxford University Press, USA, 2007.

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

[101] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A Scalable Content-Addressable Network. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pages 161–172. ACM, 2001.

[102] Sean Rhea, Dennis Geels, Timothy Roscoe, and John Kubiatowicz. Handling Churn in a DHT. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference (ATEC)*, pages 10–10, 2004.

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

[104] A. Rowstron and P. Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-scale Peer-to-Peer systems. In *Middleware*, pages 329–350. Springer, 2001.

[105] B. Russell. On the notion of cause. In *Proceedings of the Aristotelian Society*, volume 13, pages 1–26. Williams and Norgate, 1912.

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

[107] O.D. Sahin, D. Agrawal, and A. El Abbadi. Techniques for efficient routing and load balancing in content-addressable networks. In *5th IEEE International Conference on Peer-to-Peer Computing* , pages 67–74, Aug.-2 Sept. 2005.

[108] OD Sahin, A. Gulbeden, F. Emekci, D. Agrawal, and A. El Abbadi. PRISM: indexing Multi-dimensional Data in P2P Networks using Reference Vectors. In *Proceedings of the 13th Annual ACM International Conference on Multimedia*, page 955. ACM, 2005.

[109] Mario Schlosser, Michael Sintek, Stefan Decker, and Wolfgang Nejdl. HyperCuP - Hypercubes, Ontologies and Efficient Search on P2P Networks. In *LNCS*, pages 112–124. Springer, 2002.

[110] T. Schutt, F. Schintke, and A. Reinefeld. A Structured Overlay for multi-dimensional Range Queries. *Euro-Par*, pages 503–513, 2007.

[111] R. Schwarz and F. Mattern. Detecting causal relationships in distributed computations: In search of the holy grail. *Distributed Computing*, 7(3):149–174, 1994.

[112] Andy Seaborne. RDQL - A Query Language for RDF. Technical report, W3C (proposal), 2004.

[113] S. Staab and H. Stuckenschmidt. *Semantic Web and Peer-to-Peer*. Springer, 2006.

[114] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. 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)*, pages 149–160, New York, NY, USA, 2001. ACM.

[115] C. Tang and P.K. McKinley. Improving Multipath Reliability in Topology-Aware Overlay Networks. In *In proceedings of the 25th IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 82–88. IEEE, 2005.

[116] Tim Berners-Lee. Notation3 language. `http://www.w3.org/DesignIssues/Notation3`.

[117] Peter Triantafillou and Andreas Economides. Subscription Summarization: A New Paradigm for Efficient Publish/Subscribe Systems. In *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04)*, pages 562–571. IEEE Computer Society, 2004.

[118] C. Tryfonopoulos, S. Idreos, and M. Koubarakis. LibraRing: An architecture for distributed digital libraries based on DHTs. *Lecture notes in computer science*, 3652:25, 2005.

[119] R. Van Renesse. Causal Controversy at Le Mont St.-Michel. *ACM SIGOPS Operating Systems Review*, 27(2):44–53, 1993.

[120] P. Van Roy, S. Haridi, A. Reinefeld, J.B. Stefani, R. Yap, and T. Coupaye. Self Management for Large-Scale Distributed Systems: An Overview of the Selfman project. In *Formal Methods for Components and Objects*, pages 153–178. Springer, 2008.

[121] Spyros Voulgaris, Etienne Rivière, Anne-Marie Kermarrec, and Maarten Van Steen. 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.

[122] B. Yang and H. Garcia-Molina. Designing a Super-Peer Network. In *Proceedings of the 19th International Conference on Data Engineering (ICDE)*, volume 1063, pages 17–00, 2003.

[123] C. Zhang, A. Krishnamurthy, and R. Wang. Brushwood: Distributed Trees in Peer-to-Peer Systems. *Peer-to-Peer Systems IV*, 3640:47–57, 2005.

[124] C. Zhang, A. Krishnamurthy, and R.Y. Wang. Skipindex: Towards a Scalable Peer-to-Peer Index Service for High Dimensional Data. Technical report, Department of Computer Science, Princeton University, New Jersey, USA, 2004.

[125] Jing Zhou, Wendy Hall, and David De Roure. Building a Distributed Infrastructure for Scalable Triple Stores. *Journal of Computer Science and Technology*, 24(3):447–462, May 2009.

Centre de recherche INRIA Sophia Antipolis – Méditerranée
2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399