Management of Big Semantic Data

Javier D. Fernández [1,2] Mario Arias[3] Miguel A. Martínez-Prieto [1,2]

Claudio Gutiérrez [2]

[1] DataWeb Research, Department of Computer Science, University of Valladolid, Spain

[2] Department of Computer Science, University of Chile, Chile

[3] Digital Enterprise Research Institute, National University of Ireland, Galway

Chapter 4

Management of Big Semantic Data

In 2007 Jim Gray preached about the effects of the **Data Deluge** in the sciences (Hey, Tansley, and Tolle 2009). Whereas experimental and theoretical paradigms originally led science, some natural phenomena were not easily addressed by analytical models. In this scenario, computational simulation arose as a new paradigm enabling scientists to deal with these complex phenomena. Simulation produced increasing amounts of data, particularly from the use of advanced exploration instruments (large-scale telescopes, particle colliders, etc.) In this scenario, scientists were no longer interacting directly with the phenomena, but used powerful computational configurations to analyze the data gathered from simulations or captured by instruments. Sky maps built from the Sloan Digital Sky Survey observations, or the evidences found about the Higgs Boson are just two successful stories of just another paradigm, what Gray called the fourth paradigm: the eScience.

eScience sets the basis for scientific data exploration and identifies the common problems arising when dealing with data at large scale. It deals with the complexities of the whole scientific data workflow, from the data creation and capture, through the *organization* and *sharing* of these data with other scientists, to the final processing and analysis of such data. Gray linked these problems to the way in which data is encoded "because the only way that scientists are going to be able to understand that information is if their software can understand the information". In this way, *data representation* emerges as on of the key factors in the process of storing, organizing, filtering, analyzing, and visualizing data at large scale, but also for sharing and exchanging them in the distributed scientific environment.

Despite of its origins in science, the data deluge effects apply to many other fields. It is easy to find real cases of massive data sources, many of them part of our everyday lives. Common activities, such as adding new friends on social networks, sharing photographies, buying something electronically, or clicking in any result returned from a search engine, are continuously recorded in increasingly large datasets. *Data is the new "raw material of business"*.

Although business is one of the major contributors to the data deluge, there are many others players that should not go unnoticed. The Open Government movement, around the world, also are converting public administrations in massive data generators. In recent years, they have released large datasets containing educational, political, economic, criminal, census information, among many others. Besides, we are surrounded by multitude of sensors which continuously report information about temperature, pollution, energy consumption, the state of the traffic, the presence or absence of a fire, etc. Any information anywhere and in anytime is recorded in big and constantly evolving heterogeneous datasets which take part in the data deluge. If we add the scientific contributions, the datasets released by traditional and digital libraries, geographic data or collections from mass-media, we can see that the data deluge is definitely an ubiquitous revolution.

From the original eScience has evolved what has been called *data science* (Loukides 2012), a discipline that cope with this ubiquity, and basically refers to the science of transforming data in knowledge. The acquisition of this knowledge strongly depends on the existence of an effective data linkage which enables computers for integrating data from heterogeneous datasets. We bump again with the question of how information is encoded for different kinds of automatic processing.

Definitively, data and information standards are at the ground of this revolution, and due to its size, semi-automatic processing of them is essential. An algorithmic (and standardized) data encoding is crucial to enable computer exchange and understanding; for instance, this data representation must allow computers to resolve what a gene is or what a galaxy is or what a temperature measurement is (Hey, Tansley, and Tolle 2009). Nowadays, the use of graph-oriented representations and rich-semantic vocabularies are gaining momentum. On the one hand, graphs are flexible models for integrating data with different degrees of structure, but also enable these heterogeneous data to be linked in an uniform way. On the other hand, vocabularies describe what data mean. The most practical trend, in this line, suggests the use of the Resource Description Framework: RDF (Manola and Miller 2004), a standard model for data encoding and semantic technologies for publication, exchange and consumption of this **Big Semantic Data** at universal scale.

This chapter takes a guided tour to the challenges of Big Semantic Data management, and the role that it plays in the emergent **Web of Data**. Section 4.1 provides a brief overview of Big Data and its dimensions. Section 4.2 summarizes the Semantic Web foundations and introduces the main technologies used for describing and querying semantic data. These basics set the minimal background for understanding the notion of Web of Data. It is presented in Section 4.3 along with the Linked Data project and its open realization within the Linked Open Data movement. Section 4.4 characterizes the stakeholders and the main data flows performed in this Web of Data: *publication*, *exchange*, and *consumption*, defines them and delves in their potential for data interoperability, but also in the scalability drawbacks arising when Big Semantic Data must be processed and queried. Innovative compression techniques are introduced in Section 4.5, showing how the three Big Data dimensions (volume, velocity, and variety) can be successfully addressed through an integrated solution, called **HDT** (*Header-Dictionary-Triples*). Section 4.6 comprises our experimental results, showing that HDT allows scalability improvements to be achieved for storage, exchange, and query answering of such emerging data. Finally, Section 4.7 concludes and devises the potential of HDT for its progressive adoption in Big Semantic Data management.

4.1 Big Data

Much has been said and written these days about *Big Data*. News in relevant magazines (Cukier 2010; Dumbill 2012b; Lohr 2012), technical reports (Selg 2012) and white papers from leading enterprises (Dijcks 2012), some emergent research works in newly established conferences[1], disclosure books (Dumbill 2012a) and more applied ones (Marz and Warren 2013) are flooding us with numerous definitions, problems, and solutions related to Big Data. It is, obviously, a trending topic in technological scenarios, but it also is producing political, economical, and scientific impact.

We will adopt in this article a simple Big Data characterization. We refer to Big Data as "*the data that exceed the processing capacity of conventional database systems*" (Dumbill 2012b). Thus, any of those huge datasets generated in the data deluge may be considered Big Data. It is clear that they are too big, they move too fast, and they do not fit, generally, the relational model strictures (Dumbill 2012b). Under these considerations, Big Data result in the convergence of the following three "V's":

Volume is the most obvious dimension because of the large amount of data continuously gathered and stored in massive datasets exposed for different uses and purposes. *Scalability* is the main challenge related to Big Data volume by considering that effective storage mechanisms are the first requirement in this scenario. It is worth noting that storage decisions influence data retrieval, the ultimate goal for the user, that expects it to be performed as fast as possible, specially in real-time systems.

Velocity describes how data flow, at high rates, in an increasingly distributed scenario. Nowadays, velocity increases in a similar way than volume. *Streaming data processing* is the main challenge related to this dimension because selective storage is mandatory for practical volume management but also for real-time response.

Variety refers to various degrees of structure (or lack thereof) within the source data (Halfon 2012). This is mainly due to Big Data may come from multiple origins (e.g. sciences, politics, economy, social networks, or web server logs, among others) and each one describes its own semantics, hence data follow a specific structural modeling. The main challenge of Big Data variety is to achieve an effective mechanism for linking diverse classes of data differing in the inner structure.

Whereas volume and velocity address physical concerns, variety refers to a logical question mainly related to the way in which data are modeled for enabling effective integration. It

[1]Big Data conferences: http://lanyrd.com/topics/big-data/

is worth noting that the more data are integrated, the more interesting knowledge may be generated, increasing the resulting dataset value. Under these considerations, one of the main objectives in big data processing is to increase data value as much as possible by directly addressing the Big Data variety. As mentioned, the use of semantic technologies seems to be ahead in this scenario, leading to the publication of big semantic datasets.

4.2 What is Semantic Data?

Semantic data have been traditionally related to the concept of Semantic Web. The *Semantic Web* enhances the current WWW by incorporating machine-processable semantics to their information objects (pages, services, data sources, etc.). Its goals are summarized as follows:

1. *To give semantics to information on the WWW.* The difference between the approach of information retrieval techniques (that currently dominate WWW information processing) and database ones, is that in the latter data is structured via schemas, that essentially are *metadata*. Metadata gives the meaning (the semantics) to data, allowing structured query, that is, querying data with logical meaning and precision.

2. *To make semantic data on the WWW machine-processable.* Currently, on the WWW the semantics of the data is given by humans (either directly during manual browsing and searching, or indirectly via information retrieval algorithms which use human feedback entered via static links or logs of interactions). Although its current success, this process has known limitations (Quesada 2008). For big data, it is crucial to automatize the process of "understanding" (giving meaning to) data on the WWW. This amounts to develop machine-processable semantics.

To fulfill these goals, the Semantic Web community and the World Wide Consortium (W3C)[2] have developed i) models and languages for representing the semantics, and ii) protocols and languages for querying it. We will briefly describe them in the next items.

[2]http://www.w3.org

4.2.1 Describing Semantic Data

Two families of languages sufficiently flexible, distributively extensible, and machine-processable, have been developed for describing semantic data.

1. The *Resource Description Framework* (RDF) (Manola and Miller 2004). It was designed to have a simple data model, with a formal semantics, with an extensible URI-based vocabulary, and which allows anyone to distributedly make statements about any resource on the Web. In this regards, an RDF description turns out to be a set of URI triples, with the standard intended meaning. It follows the ideas of semantic networks and graph data specifications, based on universal identifiers. It gives basic tools for linking data, plus a lightweight machinery for coding basic meanings. It has two levels:

a) *Plain RDF* is the basic data model for resources and relations between them. It is based in a basic vocabulary: a set of properties, technically binary predicates. Formally it consists of triples of the form (s, p, o) (subject-predicate-object) where s, p, o are URIs that use distributed vocabularies. Descriptions are statements in the subject-predicate-object structure, where predicate and object are resources or strings. Both subject and object can be anonymous entities (blank nodes). Essentially RDF builds graphs labeled with meaning.

b) *RDFS* adds over RDF a built-in vocabulary with a normative semantics, the RDF Schema (Brickley 2004). This vocabulary deals with inheritance of classes and properties, as well as typing, among other features. It can be thought of as a lightweight ontology.

2. The Web Ontology Language (OWL) (McGuinness and van Harmelen 2004). It is a version of logic languages adapted to cope with the Web requirements, composed of basic logic operators plus a mechanism for defining meaning in a distributed fashion.

From a metadata point of view, OWL can be considered a rich vocabulary with high expressive power (classes, properties, relations, cardinality, equality, constraints, etc.). It comes in many flavours, but this gain in expressive power is at the cost of scalability (complexity of evaluation and processing). In fact, using the semantics of OWL amounts to introduce logical reasoning among pieces of data, thus exploiting in complexity terms.

4.2.2 Querying Semantic Data

If one has scalability in mind, due to complexity arguments, the expressive power of the semantics should stay at a basic level of metadata, that is *plain RDF*. This follows from the W3C design principles of interoperability, extensibility, evolution and decentralization.

As stated, RDF can be seen as a graph labeled with meaning, in which each triple (s, p, o) is represented as a direct edge-labeled graph $s \xrightarrow{p} o$. The data model RDF has a corresponding query language, called SPARQL. SPARQL (Prud'hommeaux and Seaborne 2008) is the W3C standard for querying RDF. It is essentially a graph-pattern matching query language, composed of three parts:

a) The *pattern matching part*, which includes the most basic features of graph pattern matching, like optional parts, union of patterns, nesting, filtering values of possible matchings, and the possibility of choosing the data source to be matched by a pattern.

b) The *solution modifiers* which, once the output of the pattern has been computed (in the form of a table of values of variables), allow to modify these values applying standard classical operators like projection, distinct, order and limit.

c) Finally, the *output* of a SPARQL query comes in tree forms. 1) May be: yes/no queries (ASK queries); 2) Selections of values of the variables matching the patterns (SELECT queries), and 3) Construction of new RDF data from these values, and descriptions of resources (CONSTRUCT queries).

A SPARQL query $\mathcal{Q}$ comprises head and body. The body is a complex RDF graph pattern expression comprising *triple patterns* (e.g. RDF triples in which each subject, predicate or object may be a variable) with conjunctions, disjunctions, optional parts and constraints over the values of the variables. The head is an expression that indicates how to construct the answer for $\mathcal{Q}$. The evaluation of $\mathcal{Q}$ against an RDF graph $\mathcal{G}$ is done in two steps: i) the body of $\mathcal{Q}$ is matched against $\mathcal{G}$ to obtain a set of bindings for the variables in the body, and then ii) using the information on the head, these bindings are processed applying classical relational operators (projection, distinct, etc.) to produce the answer $\mathcal{Q}$.

4.3 The Web of (Linked) Data

The WWW has enabled the creation of a global space comprising linked documents (Heath and Bizer 2011) which express information in a human-readable way. All agree that the WWW has revolutionized the way we consume information, but its document-oriented model prevents machines and automatic agents for directly accessing to the raw data underlying to any web page content. Then main reason is that documents are the atoms in the WWW model and data lack of an identity within them. This is not a new story: an "universal database", in which all data can be identified at world scale, is a cherished dream in Computer Science.

The Web of Data (Bizer, Heath, and Berners-Lee 2009) emerges under all previous considerations in order to convert raw data into first class citizens of the WWW. It materializes the Semantic Web foundations and enables raw data, from diverse fields, to be interconnected within a cloud of data-to-data hyperlinks. It achieves a ubiquitous and seamless data integration to the lowest level of granularity over the WWW infrastructure. It is worth noting that this idea does not break with the WWW as we know. It only enhances the WWW with additional standards which enable data and documents to coexist in a common space. The Web of Data grows progressively according to the Linked Data principles.

4.3.1 Linked Data

The Linked Data project[3] originated in leveraging the practice of linking data to the semantic level, following Tim Berners-Lee's ideas (Berners-Lee 2006). Its authors state that:

> *Linked Data is about using the WWW to connect related data that wasn't previously linked, or using the WWW to lower the barriers to linking data currently*

[3]http://www.linkeddata.org

linked using other methods. More specifically, Wikipedia defines Linked Data as "a term used to describe a recommended best practice for exposing, sharing, and connecting pieces of data, information, and knowledge on the Semantic Web using URIs (Uniform Resource Identifiers) and RDF".

The idea is to leverage the WWW infrastructure to produce, publish and consume data (not only documents in the form of web pages). These processes are done by different stakeholders, with different goals, in different forms and formats, in different places. One of the main challenges is the meaningful interlinking of this universe of data (Hausenblas and Karnstedt 2010). It relies on the following four rules:

1. *Use URIs as names for things.* This rule enables each possible real-world entity or its relationships to be unequivocally identified at universal scale. This simple decision guarantees any raw data has its own identity in the global space of the Web of Data.
2. *Use HTTP URIs so that people can look up those names.* This decision leverages HTTP to retrieve all data related to a given URI.
3. *When someone looks up a URI, provide useful information, using standards.* It standardizes processes in the Web of Data and pacts the languages spoken by stakeholders. RDF and SPARQL, together with semantic technologies described in the previous section, defines the standards mainly used in the Web of Data.
4. *Include links to other URIs.* It materializes the aim of data integration by simply adding new RDF triples which link data from two different datasets. This inter-dataset linkage enables the automatic browsing.

These four rules provide the basics for publishing and integrating Big Semantic Data into the global space of the Web of Data. They enable raw data to be simple encoded by combining the RDF model and URI-based identification, both for entities and for their relationships adequately labeled over rich semantic vocabularies. Tim Berners-Lee expresses as follows the Linked Data relevance (Berners-Lee 2002):

Linked Data allows different things in different datasets of all kinds to be connected. The added value of putting data on the WWW is given by the way it can be queried in combination with other data you might not even be aware of. People will be connecting scientific data, community data, social web data, enterprise data, and government data from other agencies and organizations, and other countries, to ask questions not asked before.

Linked data is decentralized. Each agency can source its own data without a big cumbersome centralized system. The data can be stitched together at the edges, more as one builds a quilt than the way one builds a nuclear power station.

A virtuous circle. There are many organizations and companies which will be motivated by the presence of the data to provide all kinds of human access to this data, for specific communities, to answer specific questions, often in connection with data from different sites.

The project and further information about linked data can be found in (Bizer, Heath, and Berners-Lee 2009; Heath and Bizer 2011).

4.3.2 Linked Open Data

Although Linked Data does not prevent its application in closed environments, (private institutional networks on any class of intranet) the most visible example of adoption and application of its principles runs openly. The Linked Open Data (LOD) movement set semantic data to be released under open licenses which do not impede data reuse for free. Tim Berners-Lee also devised a "five-stars" test to measure how these Open Data implements the Linked Data principles:

1. Make your stuff available on the web (whatever format).
2. Make it available as structured data (e.g. excel instead of image scan of a table).
3. Use non-proprietary format (e.g. CSV instead of excel).
4. Use URLs to identify things, so that people can point at your stuff.
5. Link your data to other people's data to provide context.

The LOD cloud has grown significantly since its origins in May 2007[4]. The first report pointed that 12 datasets were part of this cloud, 45 were acknowledged in September 2008, 95 datasets in 2009, 203 in 2010, and 295 different datasets in the last estimation (September 2011). These last statistics[5] point that more than 31 billion triples are currently published and more than 500 million links establish cross-relations between datasets. Government data are predominant in LOD, but other fields like geography, life sciences, media or publications are also strongly represented. It is worth emphasizing the existence of many cross-domain datasets comprising data from some diverse fields. These tend to be hubs because providing data which may be linked from and to the vast majority of specific datasets. DBpedia[6] is considered the nucleus for the LOD cloud (Auer, Bizer, Kobilarov, Lehmann, and Ives 2007). In short, DBpedia gathers raw data underlying to the Wikipedia web pages and exposes the resulting representation following the Linked Data rules. It is an interesting example of Big Semantic Data, and its management is considered within our experiments.

4.4 Stakeholders and Processes in Big Semantic Data

Although we identify data scientists as one of the main actors in the management of Big Semantic Data, we also unveil potential "traditional" users when moving from a Web of documents to a Web of data, or, in this context, to a Web of Big Semantic Data. The scalability problems arising for data experts and general users cannot be the same, as these are supposed to manage the information under different perspectives. A data scientist can make strong efforts to create novel semantic data or to analyze huge volumes of data created by third parties. She can make use of data-intensive computing, distributed machines and algorithms: to spend several hours performing a closure of a graph is perfectly accepted. In contrast, a common user retrieving, for instance, all the movies shot in New York in a given year, expects not an immediate answer, but a reasonable response time. Although one could establish an strong frontier between data (and their problems) of these worlds, we cannot forget that establishing and discovering links between diverse data is beneficial for all parties. For instance, in life sciences it is important to have links between the bibliographic data of

[4]`http://richard.cyganiak.de/2007/10/lod/`
[5]`http://www4.wiwiss.fu-berlin.de/lodcloud/state/`
[6]`http://dbpedia.org`

publications and the concrete genes studied in each publication, thus another researchers can look up previous findings of the genes they are currently studying.

The concern here is to address specific management problems while remaining in a general open representation and publication infrastructure in order to leverage the full potential of Big Semantic Data. Under this premise, a first characterization of the involved roles and processes would allow researchers and practitioners to clearly focus their efforts on a particular area. This section provides an approach toward this characterization. We first establish a simple set of stakeholders in Big Semantic Data, from where we define a common data workflow in order to better understand the main processes performed in the Web of Data.

4.4.1 Participants and Witnesses

One of the main breakthroughs after the creation of the Web, was the the consideration of the common citizen as the main stakeholder, i.e., an part involved not only in the consumption, but also in the creation of content. To emphasize this fact the notion of Web 2.0 was coined, and its implications such as blogging, tagging or social networking became one of the roots of our current sociability.

The Web of Data can be considered as a complementary dimension to this successful idea, which addresses the datasets problems of the Web. It focused on representing knowledge through machine readable descriptions (i.e. RDF), using specific languages and rules for knowledge extraction and reasoning. How this could be achieved by the general audience, and exploited for the general market, will determine its chances to success beyond the scientific community.

To date, neither the creation of self-described semantic content nor the linkage to other sources, are simple tasks for a common user. There exists several initiatives to bring semantic data creation to a wider audience, being the most feasible the use of RDFa (Adida, Herman, Sporny, and Birbeck 2012). Vocabulary and link discovery can also be mitigated through searching and recommendation tools (Volz, Bizer, Gaedke, and Kobilarov 2009; Hogan, Harth, Umbrich, Kinsella, Polleres, and Decker 2011). However, in general terms, one could

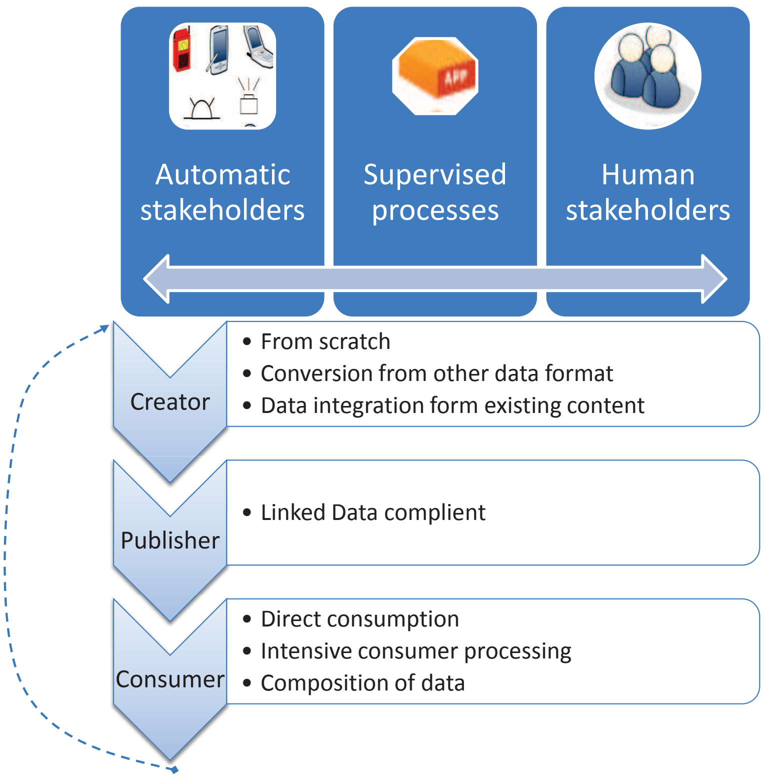

Figure 1: Stakeholder classification in Big Semantic Data management.

argue that the creation of semantic data is still almost as narrow as the original content creation in Web 1.0. In the LOD statistics, previously reported, only 0.42% of the total data is user-generated. It means that public organizations (governments, universities, digital libraries, etc.), researchers and innovative enterprises are the main creators, whereas citizens are, at this point, just witnesses of a hidden increasingly reality.

This reality shows that these few creators are able to produce huge volumes of RDF data, yet we will argue, in next section, about the quality of these publication schemes (in agreement with empirical surveys (Hogan, Umbrich, Harth, Cyganiak, Polleres, and Decker 2012)). In what follows, we characterize a minimum set of stakeholders interacting with this huge graph of knowledge with such an enormous potential. Figure 1 illustrates the main identified stakeholders within Big Semantic Data. Three main roles are present: *creators, publishers* and *consumers*, with an internal subdivision by creation method or intended use. In parallel, we distinguish between *automatic stakeholders, supervised processes*, and *human stakeholders*. We define below each stakeholder, assuming that i) this classification may not be complete as it is intended to cover the minimum foundations to understand the managing processes in Big Semantic Data and ii) categories are not disjoint; an actor could participate with several roles in a real-world scenario.

Creator: one that generates a new RDF dataset by, at least, one of these processes:

- *Creation from scratch*: the novel dataset is not based on a previous model. Even if the data exist beforehand, the data modeling process is unbiased from the previous data format. RDF authoring tools[7] are traditionally used.
- *Conversion from other data format*: the creation phase is highly determined by the conversion of the original data source; potential mappings between source and target data could be used; e.g. from relational databases (Arenas, Bertails, Prud'hommeaux, and Sequeda 2012)), as well as (semi-)automatic conversion tools[8].
- *Data integration from existing content*: the focus moves to an efficient integration of vocabularies and the validation of shared entities (Knoblock, Szekely, Ambite, Gupta, Goel, Muslea, Lerman, and Mallick 2012).

Several tasks are shared among all three processes. Some examples of this commonalities are the identification of the entities to be modeled (but this task is more important in the creation from scratch, as no prior identification has been done) or the vocabulary reuse (crucial in data integration in which different ontologies could be aligned). A complete description of the creation process is out of the scope of this work (the reader can find a guide for Linked Data creation in (Heath and Bizer 2011)).

Publisher: one that makes RDF data publicly available for different purposes and users. From now on, let us suppose that the publisher follows the Linked Data principles. We distinguish creators from publishers as, in many cases, the roles can strongly differ. Publishers do not have to create RDF content but they are responsible of the published information, the availability of the offered services (such as querying), and the correct adaptation to Linked Data principles. For instance, a creator could be a set of sensors giving the temperature in a given area in RDF (Atemezing, Corcho, Garijo, Mora, Poveda-Villalón, Rozas, Vila-Suero,

[7]A list of RDF authoring tools can be found at http://www.w3.org/wiki/AuthoringToolsForRDF
[8]A list of RDF converters can be found at http://www.w3.org/wiki/ConverterToRdf

and Villazón-Terrazas 2012), while the publisher is an entity who publish this information and provide entry points to this information.

Consumer: one that makes use of published RDF data:

- *Direct consumption*: a process whose computation task mainly involves the publisher, without intensive processing at the consumer. Downloads of the total dataset (or subparts), online querying, information retrieval, visualization or summarization are simple examples in which the computation is focused on the publisher.
- *Intensive consumer processing*: processes with a non-negligible consumer computation, such as offline analysis, data mining or reasoning over the full dataset or a subpart (live views (Tummarello, Cyganiak, Catasta, Danielczyk, Delbru, and Decker 2010)).
- *Composition of data*: those processes integrating different data sources or services, such as federated services over the Web of Data (Schwarte, Haase, Hose, Schenkel, and Schmidt 2011; Taheriyan, Knoblock, Szekely, and Ambite 2012) and RDF snippets in search engines (Haas, Mika, Tarjan, and Blanco 2011).

As stated, we make an orthogonal classification of the stakeholders attending the nature of creators, publishers and consumers. For instance, a sensor could directly create RDF data, but it could also consume RDF data.

Automatic stakeholders, such as sensors, Web processes (crawlers, search engines, recommender systems), RFID labels, smart phones, etc. Automatic RDF streaming, for instance, would become a hot topic, specially within the development of smart cities (De, Elsaleh, Barnaghi, and Meissner 2012). Note that, although each piece of information could be particularly small, the whole system can be seen also as a big semantic dataset.

Supervised processes, i.e. processes with human supervision, as semantic tagging and folksonomies within social networks (García-Silva, Corcho, Alani, and Gómez-Pérez 2012).

Human stakeholders, who perform most of the task for creating, publishing or consuming RDF data.

The following running example provides a practical review of this classification. Nowadays, an RFID tag could document a user context through RDF metadata descriptions (Foulonneau 2011). We devise a system in which RFID tags provide data about temperature and position. Thus, we have thousands of sensors providing RDF excerpts modeling the temperature in distinct parts of a city. Users can visualize and query online this information, establishing some relationships for example with special events (such as a live concert or sport matches). In addition, the RDF can be consumed by a monitoring system, e.g. to alert the population in case of extreme temperatures.

Following the classification, each sensor is an automatic creator, conforming all together a potentially huge volume of RDF data. Whereas a sensor should be designed to take care of RDF description (e.g. to follow a set of vocabularies and description rules and to minimize the size of descriptions), it can not address publishing facilities (query endpoints, services to user, etc.). Alternatively, intermediate hubs would collect the data and the authoritative organization will be responsible of its publication, and applications and services over these data. This publication authority would be considered as a supervised process solving scalability issues of huge RDF datastreams for collecting the information, filtering it (e.g. eliminating redundancy) and finally complying with Linked Data standards. Although these processes could be automatic, let us suppose that human intervention is needed to define links between data, for instance linking positions to information about city events. Note also that intermediate hubs could be seen as supervised consumers of the sensors, yet the information coming from the sensors is not openly published but streamed to the appropriate hub. Finally, the consumers are humans, in case of the online users (concerned of query resolution, visualization, summarization, etc.) or an automatic (or semi-atomatic) process, in case of monitoring (doing potential complex inference or reasoning).

4.4.2 The Workflow of Publication-Exchange-Consumption

The previous RFID network example shows the enormous diversity of processes and different concerns for each type of stakeholder. In what follows, we will consider the creation step out of the scope of this work, because our approach relies on the existence of big RDF datasets (without belittling those ones which can be created hereinafter). We focus on tasks involving large-scale management; for instance, scalability issues of visual authoring a big

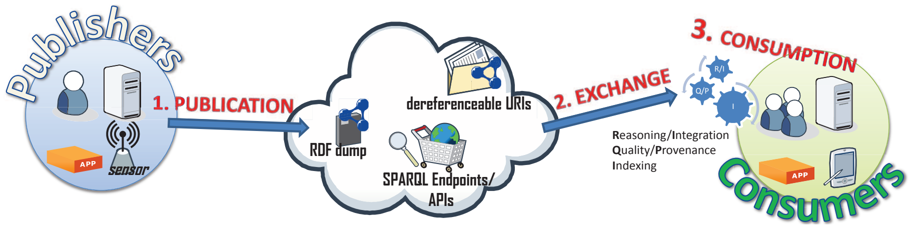

Figure 2: Publication-Exchange-Consumption workflow in the Web of Data.

RDF dataset are comparable to RDF visualization by consumers, or the performance of RDF data integration from existing content depends on efficient access to the data and thus existing indexes, a crucial issue also for query response.

Management processes for publishers and consumers are diverse and complex to generalize. However, it is worth characterizing a common workflow present in almost every application in the Web of Data in order to place scalability issues in context. Figure 2 illustrates the identified workflow of Publication-Exchange-Consumption.

Publication refers to the process of making RDF data publicly available for diverse purposes and users, following the Linked Data principles. Strictly, the only obligatory "service" in these principles is to provide dereferenceable URIs, i.e., related information of an entity. In practice, publishers complete this basic functionality exposing their data through public APIs, mainly via SPARQL endpoints, a service which interprets the SPARQL query language. They also provide RDF dumps, files to fully or partly download the RDF dataset.

Exchange is the process of information exchange between publishers and consumers. Although the information is represented in RDF, note that consumers could obtain different "views" and hence formats, some of then not necessarily in RDF. For instance, the result of a SPARQL query could be provided in a CSV file or the consumer would request a summary with statistics of the dataset in a XML file. As we are issuing management of semantic datasets, we restrict exchange to RDF interchange. Thus we rephrase exchange as the process of RDF exchange between publishers and consumers after an RDF dump request, a SPARQL query resolution or another request or service provided by the publisher.

Consumption can involve, as stated, a wide range of processes, from direct consumption to intensive processing and composition of data sources. Let us simply define the consumption as the use of potentially large RDF data for diverse purposes.

A final remark must be done. The workflow definition seems to restrict the management to large RDF datasets. However, we would like to open scalability issues to a wider range of publishers and consumers with more limited resources. For instance, similar scalability problems arise when managing RDF in mobile devices; although the amount of information could be potentially smaller, these devices have more restrictive requirements for transmission costs/latency, and for post-processing due to their inherent memory and CPU constraints (Le-Phuoc, Parreira, Reynolds, and Hauswirth 2010). In the following, whenever we provide approaches for managing these processes in large RDF datasets, we ask the lecture to take into consideration this appreciation.

4.4.3 State-of-the-art for Publication-Exchange-Consumption

This section summarizes some of the current trends to address publication, exchange and consumption at large scale.

Publication schemes: the straightforward publication, following Linked Data principles, presents several problems in large datasets (Fernández, Martínez-Prieto, and Gutiérrez 2010); a previous analysis of published RDF datasets reveals several undesirable features; the provenance and metadata about contents are barely present, and their information is neither complete nor systematic. Furthermore, the RDF dump files have neither internal structure nor a summary of their content. A massive empirical study of Linked Open Data datasets in (Hogan, Umbrich, Harth, Cyganiak, Polleres, and Decker 2012) draws similar conclusions; few providers attach human readable metadata to their resources or licensing information. Same features can be applied to SPARQL endpoints, in which a consumer knows almost nothing about the content she is going to query beforehand. In general terms, except for the general Linked Data recommendations (Heath and Bizer 2011), few works address the publication of RDF at large scale.

The Vocabulary of Interlinked Datasets: VoiD (Alexander, Cyganiak, Hausenblas, and Zhao 2009) is the nearest approximation to the discovery problem, providing a bridge between publishers and consumers. Publishers make use of a specific vocabulary to add metadata to their datasets, e.g. to point to the associated SPARQL endpoint and RDF dump, to describe the total number of triples and to connect to linked datasets. Thus, consumers can look up this metadata to discover datasets or to reduce the set of interesting datasets in federated queries over the Web of Data (Akar, Hala, Ekinci, and Dikenelli 2012). Semantic Sitemaps (Cyganiak, Stenzhorn, Delbru, Decker, and Tummarello 2008) extends the traditional Sitemap Protocol for describing RDF data. They include new XML tags so that crawling tools (such as Sindice[9]) can discover and consume the datasets.

As a last remark, note that deferenceable URIs can be done in a straightforward way, publishing one document per URI, or set of URIs. However, the publisher commonly materializes the output by querying the dataset at URI resolution time. This moves the problem to the underneath RDF store, which has also to deal with scalability problems (see "Efficient RDF Consumption" below). The empirical study in (Hogan, Umbrich, Harth, Cyganiak, Polleres, and Decker 2012) also confirmed that publishers often do not provide locally-known inlinks in the dereferenced response which must be taken into account by consumers.

RDF Serialization Formats. As we previously stated, we focus on exchanging large-scale RDF data (or smaller volumes in limited resources stakeholders). Under this consideration, the RDF serialization format directly determines the transmission costs and latency for consumption. Unfortunately, datasets are currently serialized in plain and verbose formats such as RDF/XML (Beckett 2004) or Notation3: N3 (Berners-Lee 1998), a more compact and readable alternative. Turtle (Beckett and Berners-Lee 2008) inherits N3 compact ability adding interesting extra features, *e.g.* abbreviated RDF datasets. RDF/JSON (Alexander 2008) has the advantage of being coded in a language easier to parse and more widely accepted in the programming world. Although all these formats present features to "abbreviate" constructions, they are still dominated by a document-centric and human-readable view which adds an unnecessary overhead to the final dataset representation.

In order to reduce exchange costs and delays on the network, universal compressors (*e.g.* gzip) are commonly used over these plain formats. In addition, specific interchange oriented representations may be also used. For instance, the Efficient XML Interchange Format: EXI (Schneider and Kamiya 2011) may be used for representing any valid RDF/XML dataset.

[9]http://sindice.com/

Efficient RDF Consumption. The aforementioned variety of consumer tasks hinders to achieve a one-size-fits-all technique. However, some general concerns can be outlined. In most scenarios, the performance is influenced by i) the serialization format, due to the overall data exchange time, and ii) the RDF indexing/querying structure. In the first case,if a compressed RDF has been exchanged, a previous decompression must be done. In this sense, the serialization format affects the consumption through the transmission cost, but also with the easiness of parsing. The latter factor affects the consumption process in different ways:

- For SPARQL endpoints and dereferenceable URIs materialization, the response time depends on the efficiency of the underlying RDF indexes at the publisher.
- Once the consumer has the dataset, the most likely scenario is indexing it in order to operate with the RDF graph, e.g. for intensive operation of inference, integration, etc.

Although the indexing at consumption could be performed once, the amount of resources required for it may be prohibitive for many potential consumers (specially for mobile devices comprising a limited computational configuration). In both cases, for publishers and consumers, an RDF store indexing the datasets is the main actor for efficient consumption.

Diverse techniques provide efficient RDF indexing, but there are still workloads for scalable indexing and querying optimization (Sidirourgos, Goncalves, Kersten, Nes, and Manegold 2008; Schmidt, Meier, and Lausen 2010). On the one hand, some RDF stores are built over relational databases and perform SPARQL queries through SQL, *e.g.* Virtuoso[10]. The most successful relational-based approach performs a vertical-partitioning, grouping triples by predicate and storing them in independent 2-column tables (S,O) (Sidirourgos, Goncalves, Kersten, Nes, and Manegold 2008; Abadi, Marcus, Madden, and Hollenbach 2009). On the other hand, some stores: Hexastore (Weiss, Karras, and Bernstein 2008) or RDF-3X (Neumann and Weikum 2010) build indexes for all possible combinations of elements in RDF (SPO, SOP, PSO, POS, OPS, OSP), allowing i) all triple patterns to be directly resolved in the corresponding index, and ii) the first join step to be resolved through fast merge-join. Although it achieves a global competitive performance, the index replication largely increases spatial requirements. Other solutions take advantage of structural properties of

[10]`http://www.openlinksw.com/dataspace/dav/wiki/Main/VOSRDF`

the data model (Tran, Ladwig, and Rudolph 2012), introduce specific graph compression techniques (Atre, Chaoji, Zaki, and Hendler 2010; Álvarez-García, Brisaboa, Fernández, and Martínez-Prieto 2011), or use distributed nodes within a Map-Reduce infrastructure (Urbani, Maassen, and Bal 2010).

4.5 An Integrated Solution for Managing Big Semantic Data

When dealing with Big Semantic Data, each step in the workflow must be designed to address the three Big Data dimensions. Whereas variety is managed through semantic technologies, this decision determines the way volume and velocity are addressed. As previously discussed, data serialization has a big impact on the workflow, as traditional RDF serialization formats are designed to be human readable instead of machine processable. They may fit smaller scenarios in which volume or velocity are not an issue, but under the presented premises, it clearly becomes a bottleneck of the whole process. We present, in the following, the main requirements for an RDF serialization format of Big Semantic Data.

- **It must be generated efficiently from another RDF input format.** For instance, a data creator having the dataset in a semantic database must be able to dump it efficiently into an optimized exchange format.
- **It must be space efficient.** The generated dump should be as small as possible, introducing compression for space savings. Bear in mind, that big semantic datasets are shared on the Web of Data, and they may be transferred through the network infrastructure to hundreds or even thousands of clients. Reducing size will not only minimize the bandwidth costs of the server, but also the waiting time of consumers that are retrieving the dataset for any class of consumption.
- **It must be ready to post-process.** A typical case is performing a sequential triple-to-triple scanning for any post-processing task. This can seem trivial, but is clearly time consuming when Big Semantic Data is post-processed at the consumer. As show in our experiments, just parsing a dataset of 640 million triples, serialized in NTriples and gzip-compressed, wastes more than 40 minutes on a modern computational configuration.

- **It must be easy to convert to other representations.** The most usual scenario at consumption involves loading the dataset into an RDF Store. Most of the solutions reviewed in the previous section use disk-resident variants of B-Trees, which keep a subset of the pages in main memory. For instance, if data is already sorted, this process is more efficient than doing it on unsorted elements. Therefore having the data pre-sorted can be a step ahead in these cases. Also, many stores keep several indexes for the different triples orderings (SPO, OPS, PSO...). If the serialization format enables data traversing to be performed in different orders, the multi-index generation process can be completed more efficiently.

- **It should be able to locate pieces of data within the whole dataset.** It is desirable to avoid a full scan over the dataset just to locate a particular piece of data. Note that this scan is a highly time consuming process in Big Semantic Data. Thus, the serialization format must retain all possible clues enabling direct access to any piece of data in the dataset. As explained in the SPARQL query language, a basic way of specifying which triples to fetch is specifying a triple pattern where each component is either a constant or a variable. A desirable format should be ready to solve most of the combinations of triple patterns (possible combinations of constants or variables in subject, predicates and objects). For instance, a typical triple pattern is to provide a subject, leaving the predicate and object as variables (and therefor the expected result). In such case, we pretend to locate all the triples that talk about a specific subject. In other words, this requirement contains a succinct intention; data must be encoded in such a way that "the data are the index".

4.5.1 Encoding Big Semantic Data: HDT

Our approach, **HDT**: *Header-Dictionary-Triples* (Fernández, Martínez-Prieto, and Gutiérrez 2010), considers all the previous requirements, addressing a machine-processable RDF serialization format which enables Big Semantic Data to be efficiently managed within the common workflows of the Web of Data. The format formalizes a compact binary serialization optimized for storage or transmission over a network. It is worth noting that HDT is described and proposed for standardization as W3C Member Submission (Fernández, Martínez-Prieto, Gutiérrez, and Polleres 2011). In addition, a succinct data structure has been proposed (Martínez-Prieto, Arias, and Fernández 2012) to browse HDT-encoded datasets. This structure holds the compactness of such representation and provides direct access to any piece of data as described below.

HDT organizes Big Semantic Data in three logical components (*Header*, *Dictionary*, and *Triples*) carefully described to address RDF peculiarities but also considering how these data are actually used in the Publication-Exchange-Consumption workflow.

Header. The Header holds, in plain RDF format, metadata describing a big semantic dataset encoded in HDT. It acts as an entry point for a consumer, who can peek on certain key properties of the dataset to have an idea of its content, even before retrieving the whole dataset. It enhances the VoID Vocabulary (Alexander, Cyganiak, Hausenblas, and Zhao 2009) to provide a standardized binary dataset description in which some additional HDT-specific properties are appended[11]. The Header component comprises four distinct sections:

- **Publication Metadata** provides information about the publication act, for instance when was the dataset generated, when was it made public, who is the publisher, where is the associated SPARQL endpoint, etc. Many properties of this type are described using the popular Dublin Core Vocabulary[12].
- **Statistical Metadata** provides statistical information about the dataset, such as the number of triples, the number of different subjects, predicates, objects, or even histograms. For instance, this class of metadata is very valuable or visualization software or federated query evaluation engines.
- **Format Metadata** describes how Dictionary and Triples components are encoded. This allows to have different implementations or representations of the same data in different ways. For instance one could prefer to have the triples in SPO order whereas other applications might need it in OPS. Also the dictionary could apply a very aggressive compression technique to minimize the size as much as possible, whereas other implementation could be focused on query speed and even include a full-text index to

[11]http://www.w3.org/Submission/2011/SUBM-HDT-Extending-VoID-20110330/
[12]http://dublincore.org/

accelerate text searches. These metadata enable the consumer for checking how an HDT-encoded dataset can be accessed in the data structure.

- **Additional Metadata**. Since the Header contains plain RDF, the publisher can enhance it using any vocabulary. It allows specific dataset/application metadata to be described. For instance, in life sciences a publisher might want to describe, in the Header, that the dataset describes a specific class of proteins.

Since RDF enables data integration at any level, the Header component ensures that HDT-encoded datasets are not isolated and can be interconnected. For instance, it is a great tool for query syndication. A syndicated query engine could maintain a catalog composed by the Headers of different HDT-encoded datasets from many publishers and use it to know where to find more data about a specific subject. Then, at query-time, the syndicated query engine can either use the remote SPARQL endpoint to query directly the third-party server, or even download the whole dataset and save it in a local cache. Thanks to the compact size of HDT-encoded datasets, both the transmission and storage costs are highly reduced.

Dictionary. The Dictionary is a catalog comprising all the different terms used in the dataset, such as URIs, literals and blank nodes. A unique identifier (ID) is assigned to each term, enabling triples to be represented as tuples of three IDs which, respectively, reference the corresponding terms in the dictionary. This is a first step toward compression, since it avoids long terms to be repeatedly represented. This way, each term occurrence is now replaced by its corresponding ID, which encoding requires less bits in the vast majority of the cases. Furthermore, the catalog of terms within the dictionary may be encoded in many advanced ways focused on boosting querying or reducing size. A typical example is to use any kind of differential compression for encoding terms sharing long prefixes, e.g. URIs.

The dictionary is divided into sections depending on whether the term plays subject, predicate, or object roles. Nevertheless, in semantic data is quite common that a URI appears both as a subject in one triple and as object on another. To avoid repeating those terms twice in the subjects and in the objects sections, we can extract them into a fourth section called *shared Subject-Object*.

Figure 3 depicts the 4-section dictionary organization and how IDs are assigned to the

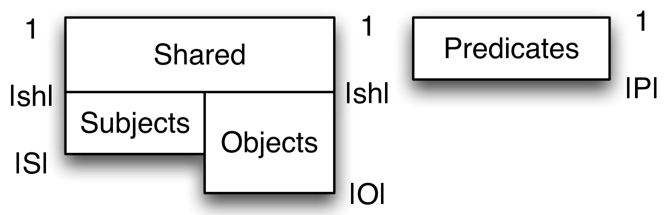

Figure 3: HDT dictionary organization into four sections.

corresponding terms. Each section is sorted lexicographically and then correlative IDs are assigned to each terms from 1 to n. It is worth noting that, for subjects and objects, the shared Subject-Object section uses the lower range of IDs; e.g. if there are m terms playing interchangeably as subject and object, all IDs x such that $x < m$ belong to this shared section.

HDT allows to use different techniques of dictionary representation. Each one can handle its catalog of terms in different ways, but must always implement these basic operations:

- `locate(term)`: finds the term and returns its ID.
- `extract(id)`: extracts the term associated to the ID.
- `numElements()`: returns the number of elements of the section.

More advanced techniques might also provide these optional operations:

- `prefix(p)`: finds all terms starting with the prefix 'p'.
- `suffix(s)`: finds all terms ending with the suffix 's'.
- `substring(s)`: finds all the terms containing the substring 's'.
- `regex(e)`: finds all strings matching the specified regular expression 'e'.

For instance, these advanced operations are very convenient when serving query suggestions to the user, or when evaluating SPARQL queries that include REGEX filters.

We suggest a Front-Coding (Witten, Moffat, and Bell 1999) based representation as the most simple way of dictionary encoding. It has been successfully used in many WWW-based applications involving URL management. It is a very simple yet effective technique based on differential compression. This technique applies to lexicographically sorted dictionaries by dividing them into buckets of b terms. By tweaking this bucket size, different space/time tradeoffs can be achieved. The first term in the bucket is explicitly stored and the remaining $b - 1$ ones are encoded with respect to their precedent: the common prefix length is first encoded and the remaining suffix is appended. More technical details about these dictionaries are available in (Brisaboa, Cánovas, Claude, Martínez-Prieto, and Navarro 2011).

The work of (Martínez-Prieto, Fernández, and Cánovas 2012) surveys the problem of encoding compact RDF dictionaries. It reports that Front-Coding achieves a good performance for a general scenario, but more advanced techniques can achieve better compression ratios and/or handle directly complex operations. In any case, HDT is flexible enough to support any of these techniques, allowing stakeholders to decide which configuration is better for their specific purposes.

Triples. As stated, the Dictionary component allows spatial savings to be achieved, but it also enables RDF triples to be compactly encoded, representing tuples of three IDs referring the corresponding terms in the Dictionary. Thus, our original RDF graph is now transformed into a graph of IDs which encoding can be carried out in a more optimized way.

We devise a Triples encoding that organizes internally the information in a way that exploits graph redundancy to keep data compact. Moreover, this encoding can be easily mapped into a data structure that allows basic retrieval operations to be performed efficiently.

Triple patterns are the SPARQL query atoms for basic RDF retrieval. That is, all triples matching a template (s, p, o) (where s, p and o may be variable) must be directly retrieved from the Triples encoding. For instance, in the geographic dataset Geonames[13], the triple pattern below searches all the subjects whose feature code (the predicate) is 'P' (the object), a shortcode for "country". In other words, it asks about all the URIs representing countries:

```
? <http://www.geonames.org/ontology#featureCode> <http://www.geonames.org/ontology#P>
```

[13]http://www.geonames.org

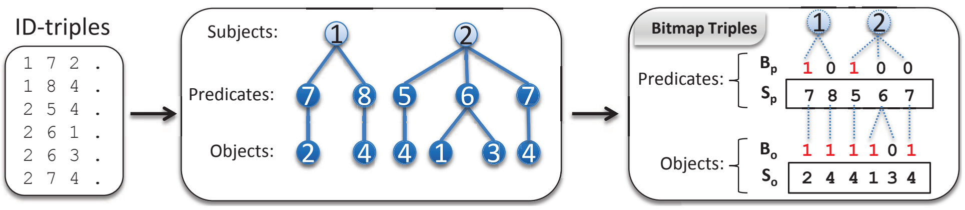

Figure 4: Description of Bitmap Triples.

Thus, the Triples component must be able to retrieve the subject of all those triples matching this pair of predicate and object.

HDT proposes a Triples encoding named *BitmapTriples* (BT). This technique needs the triples to be previously sorted in a specific order, such as subject-predicate-object (SPO). BT is able to handle all possible triple orderings, but we only describe the intuitive SPO order for explanation purposes.

Basically, BT transforms the graph into a forest containing as many trees as different subjects are used in the dataset, and these trees are then ordered by subject ID. This way, the first tree represents all triples rooted by the subject identified as `1`, the second tree represents all triples rooted by the subject identified as `2`, and so on. Each tree comprises three levels: the root represents the subject, the second level lists all predicates related to the subject, and finally the leaves organize all objects for each pair *(subject, predicate)*. Predicate and object levels are also sorted:

- All predicates related to the subject are sorted in increasing way. As Figure 4 shows, predicates are sorted as `{5,6,7}` for the second subject.
- Objects follow an increasing order for each path in the tree. That is, objects are internally ordered for each pair *(subject, predicate)*. As Figure 4 shows, the object `5` is listed first (because it is related to the pair `(2,5)`), then `1,3` (by considering that these are related to the pair `(2,6)`), and `4` is the last object because of its relation to `(2,7)`.

Each triple in the dataset is now represented as a full path root-to-leave in the corresponding tree. This simple reorganization reveals many interesting features:

- The subject can be implicitly encoded given that the trees are sorted by subject and we know the total number of trees. Thus, BT does not perform a triples encoding, but it represents pairs (predicate, object). This is an obvious spatial saving.
- Predicates are sorted within each tree. This is very similar to a well-known problem: posting list encoding for Information Retrieval (Witten, Moffat, and Bell 1999; Baeza-Yates and Ribeiro-Neto 2011). This allows applying many existing and optimized techniques to our problem. Besides, efficient search within predicate lists is enabled by assuming that the elements follow a known ordering.
- Objects are sorted within each path in the tree, so *i)* these can be effectively encoded, and *ii)* these can also be efficiently searched.

BT encodes the Triples component level-by-level. That is, predicate and object levels are encoded in isolation. Two structures are used for predicates: *i)* an ID sequence ($\mathcal{S}_p$) concatenates predicate lists following the tree ordering; *ii)* a bitsequence ($\mathcal{B}_p$) uses one bit per element in $\mathcal{S}_p$: `1` bits mean that this predicate is the first one for a given tree, whereas `0` bits are used for the remaining predicates. Object encoding is performed in a similar way: $\mathcal{S}_o$ concatenates object lists, and $\mathcal{B}_o$ tags each position in such way that `1` bits represent the first object in a path, and `0` bits the remaining ones. The right part of the Figure 4 illustrates all these sequences for the given example.

4.5.2 Querying HDT-encoded datasets: HDT-FoQ

An HDT-encode dataset can be directly accessed once its components are loaded into the memory hierarchy of any computational system. Nevertheless, this can be tuned carefully by considering the volume of the datasets and the retrieval velocity needed by specific applications. Thus, we require a data structure that keeps the compactness of the encoding to load data at the higher levels of the memory hierarchy. Data in faster memory always means faster retrieval operations. We call this solution **HDT-FoQ**: *HDT Focused on Querying.*

Dictionary. The dictionary component must be able to be directly mapped from the encoding to the computer because it must embed enough information to resolve the basic operations previously described. Thus, this component follows the idea of "the data are the index". We invite interested readers to review the paper of (Brisaboa, Cánovas, Claude, Martínez-Prieto, and Navarro 2011) for a more detailed description on how dictionaries provide indexing capabilities.

Triples. The previously described BitmapTriples approach is easy to map due to the simplicity of its encoding. Sequences $\mathcal{S}_p$ and $\mathcal{S}_o$ are loaded into two integer arrays using respectively $log(|P|)$ and $log(|O|)$ bits per element. Bitsequences can also be mapped directly, but in this case they are enhanced with an additional small structure (González, Grabowski, Mäkinen, and Navarro 2005) that ensures constant time resolution for some basic bit-operations.

This simple idea enables efficient traversal of the Triples component. All these algorithms are described in (Martínez-Prieto, Arias, and Fernández 2012), but we review them in practice over the example in Figure 4. Let us suppose that we ask for the existence of the triple `(2,6,1)`. It implies that the retrieval operation is performed over the second tree:

1. We retrieve the corresponding predicate list. It is the 2^{nd} one in $\mathcal{S}_p$ and it is found by simply locating where is the second `1` bit in $\mathcal{B}_p$. In this case $P_2 = 3$, so the predicate list comprises all elements from $\mathcal{S}_p[2]$ until the end (because no more this is the last `1` bit in $\mathcal{B}_p$). Thus, the predicate list is {`5,6,7`}.
2. The predicate `6` is searched in the list. We binary search it and find that it is the second element in the list. Thus, it is at position $P_2 + 2 - 1 = 3 + 2 - 1 = 4$ in $\mathcal{S}_p$, so we are traversing the 4^{th} path of the forest.
3. We retrieve the corresponding object list. It is the 4^{th} one in $\mathcal{S}_o$. We obtain it as before: firstly locate the fourth `1` bit in $\mathcal{B}_o$: $O_4 = 4$ and then retrieves all objects until the next `1` bit. That is, the list comprises the objects {`1,3`}.
4. Finally, the object list is binary searched and locates the object 3 in its first position. Thus, we are sure that the triple `(2,6,1)` exist in the dataset.

All triple patterns providing the subjects are efficiently resolved on variants of this process. Thus, the data structure directly mapped from the encoding provides fast subject-based retrieval, but makes difficult accessing by predicate and object. Both ones can be easily accomplished with a limited overhead on the space used by the original encoding. All fine-grain details about the following decisions are also explained in (Martínez-Prieto, Arias, and Fernández 2012).

Enabling access by predicate. This retrieval operation demands direct access to the second level of the tree, so it means efficient access to the sequence $\mathcal{S}_p$. However, the elements of $\mathcal{S}_p$ are sorted by subject, so locating all predicate occurrences demands a full scanning of this sequence and this result in a poor response time.

Although, accesses by predicate are uncommon in general (Arias, Fernández, and Martínez-Prieto 2011), some applications could require them (e.g. extracting all the information described with a set of given predicates). Thus, we must address it by considering the need of another data structure for mapping $\mathcal{S}_p$. It must enable efficient predicate locating but without degrading basic access because it is used in all operations by subject. We choose a structure called wavelet tree.

The *wavelet tree* (Grossi, Gupta, and Vitter 2003) is a succinct structure which reorganizes a sequence of integers, in a range $[1, n]$, to provide some access operations to the data in logarithmic time. Thus, the original $\mathcal{S}_p$ is now loaded as a wavelet tree, not as an array. It means a limited additional cost (in space) which holds HDT scalability for managing Big Semantic Data. In return, we can locate all predicate occurrences in logarithmic time with the number of different predicates used for modeling in the dataset. In practice, this number is small and it means efficient occurrences location within our access operations. It is worth noting that to access to any position in the wavelet tree has also now a logarithmic cost.

Therefore, access by predicate is implemented by firstly performing an occurrence-to-occurrence location, and for each one traversing the tree by following comparable steps to than explained in the previous example.

Enabling access by object. The data structure designed for loading HDT-encoded datasets, considering a subject-based order, is not suitable for doing accesses by object. All the occurrence of an object are scattered throughout the sequence $\mathcal{S}_o$ and we are not able to locate them unless we do sequential scan. Furthermore, in this case an structure like the Wavelet Tree becomes inefficient; RDF datasets usually have few predicates, but they contain many different objects and logarithmic costs result in very expensive operation.

We enhance HDT-FoQ with an additional index (called O-Index), that is responsible for solving accesses by object. This index basically gathers the positions in where each object appears in the original $\mathcal{S}_o$. Please note, that each leave is associated to a different triple, so given the index of an element in the lower level, we can guess the predicate and subject the associated by traversing the tree upwards using the bitsequences as previously.

In relative terms, this O-Index has a significant impact in the final HDT-FoQ requirements because it takes considerable space in comparison to the other data structures used for modeling the Triples component. However, in absolute terms, the total size required by HDT-FoQ is very small in comparison to that required by the other competitive solutions in the state-of-the-art. All these results are analyzed in the next section.

Joining Basic Triple Patterns. All this infrastructure enables basic triple patterns to be resolved, in compressed space, at higher levels of the hierarchy of memory. As we show below, it guarantees efficient triple pattern resolution. Although this kind of queries are massively used in practice (Arias, Fernández, and Martínez-Prieto 2011), the SPARQL core is defined around the concept of Basic Graph Pattern (BGP) and its semantics to build conjunctions, disjunctions, and optional parts involving more than a single triple pattern. Thus, HDT-FoQ must provide more advanced query resolution to reach a full SPARQL coverage. At this moment, it is able to resolve conjunctive queries by using specific implementations of the well-known *merge* and *index* join algorithms (Ramakrishnan and Gehrke 2000).

4.6 Experimental Results

This section analyzes the impact of HDT for encoding Big Semantic Data within the Publication-Exchange-Consumption workflow described in the Web of Data. We characterize the publisher and consumer stakeholders of our experiments as follows:

- The **publisher** is devised as an efficient agent implementing a powerful computational configuration. It runs on an Intel Xeon E5645@2.4GHz, hexa-core (6cores-12siblings:

Table 1: Statistics of the real-world datasets used in the experimentation.

Dataset	Plain Ntriples	Size (GB)	Available at
LinkedMDB	6,148,121	0,85	`http://queens.db.toronto.edu/~oktie/linkedmdb`
DBLP	73,226,756	11,16	`http://DBLP.l3s.de/DBLP++.php`
Geonames	119,316,724	13,79	`http://download.Geonames.org/all-Geonames-rdf.zip`
DBpedia	296,907,301	48,62	`http://wiki.dbpedia.org/Downloads37`
Freebase	639,078,932	84,76	`http://download.freebase.com/datadumps/`[14]
Mashup	1,055,302,957	140,46	`Mashup of Geonames+Freebase+DBPedia`

2 thread per core), 96GB DDR3@1066Mhz.

- The **consumer** is designed on a conventional configuration because it plays the role of any agent consuming RDF within the Web of Data. It runs on an AMD-PhenomTM-II X4 955@3.2GHz, quad-core (4cores-4siblings: 1thread per core), 8GB DDR2@800MHz.

The **network** is regarded as an ideal communication channel: free of errors and any other external interferences. We assume a transmission speed of 2Mbyte/s.

All our experiments are carried out over an heterogeneous data configuration of many colors and flavors. We choose a variety of real-world semantic datasets of different sizes and from different application domains (see Table 1). In addition, we join together the three bigger datasets into a large `mashup` of more than 1 billion triples to analyze performance issues in an integrated dataset.

The prototype running these experiments is developed in C++ using the HDT library publicly available at the official RDF/HDT website[15].

4.6.1 Publication Performance

As explained, RDF datasets are usually released in plain-text form (NTriples, Turtle, or RDF-XML), and their big volume is simply reduced using any traditional compressor. This way, volume directly affects the publication process because the publisher must, at least, process the dataset to convert it to a suitable format for exchange. Attending to the current practices, we set `gzip` compression as the baseline and we also include `lzma` because of its

[15] `http://www.rdfhdt.org`

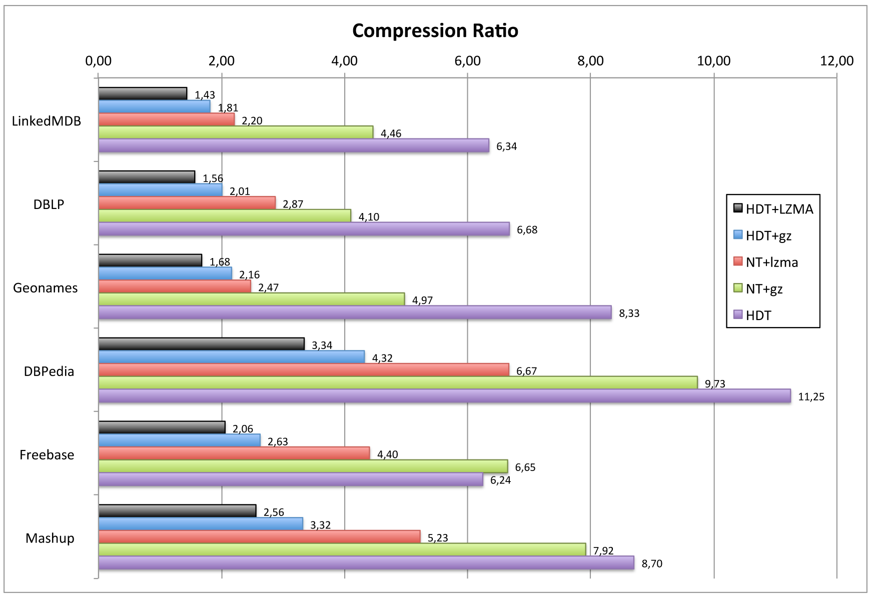

Figure 5: Dataset compression (expressed as percent of the original size in NTriples).

effectiveness. We compare their results against HDT, in plain and also in conjunction with the same compressors. That is, `HDT plain` implements the encoding described in Section 4.5.1, and `HDT+X` stands for the result of compressing `HDT plain` with the compressor `X`.

Figure 5 shows compression ratios for all the considered techniques. In general, `HDT plain` requires more space than traditional compressors. It is an expected result because both Dictionary and Triples use very basic approaches. Advanced techniques for each component enable significant improvements in space. For instance, our preliminary results using the technique proposed in (Martínez-Prieto, Fernández, and Cánovas 2012) for dictionary encoding, show a significant improvement in space. Nevertheless, if we apply traditional compression over the HDT-encoded datasets, the spatial requirements are largely diminished. As shown in Figure 5, the comparison changes when the HDT-encoded datasets are compressed with `gzip` and `lzma`. These results show that `HDT+lzma` achieves the most compressed representations, largely improving the effectiveness reported by traditional approaches. For instance, `HDT+lzma` only uses 2.56% of the original `mashup` size, whereas compressors require 5.23% (`lzma`) and 7.92% (`gzip`).

Thus, encoding the original Big Semantic Data with HDT and then applying compression reports the best numbers for publication. It means that publishers using our approach

Table 2: Publication times (minutes).

Dataset	gzip	lzma	HDT+ gzip	HDT+ lzma
LinkedMDB	**0.19**	14.71	1.09	1.52
DBLP	**2.72**	103.53	13.48	21.99
Geonames	**3.28**	244.72	26.42	38.96
DBPedia	**18.90**	664.54	84.61	174.12
Freebase	**24.08**	1154.02	235.83	315.34
Mashup	**47.23**	2081.07	861.87	1033.03

require $2-3$ times less storage space and bandwidth than using traditional compression. These savings are achieved at the price of spending some time to obtain the corresponding representations. Note that traditional compression basically requires compressing the dataset, whereas our approach firstly transforms the dataset into its HDT encoding and then compresses it. These *publication times* (in minutes) are depicted in Table 2.

As can be seen, direct publication, based on `gzip` compression, is up to 20 times faster than `HDT+gzip`. The difference is slightly higher compared to `HDT+lzma`, but this choice largely outperforms direct `lzma` compression. However, this comparison must be carefully analyzed because publication is a batch process and it is performed only once per dataset, whereas exchange and post-processing costs are paid each time that any consumer retrieves the dataset. Thus, in practical terms, publishers will prioritize compression versus publication time because: i) storage and bandwidth savings, and ii) the overall time that consumers wait when they retrieve the dataset.

4.6.2 Exchange Performance

In the ideal network regarded in our experiments, exchange performance is uniquely determined by the data size. Thus, our approach also appears as the most efficient because of it excellent compression ratios. Table 3 organizes processing times for all datasets and each task involved in the workflow. Column `exchange` lists exchanging times required when `lzma` (in the baseline) and `HDT+lzma` are used for encoding.

For instance, the `mashup` exchange takes roughly half an hour for `HDT+lzma` and slightly more than one hour for `lzma`. Thus, our approach reduces by the half exchange time and also saves bandwidth in the same proportion for the `mashup`.

4.6.3 Consumption Performance

In the current evaluation, consumption performance is analyzed from two complementary perspectives. First, we consider a *post-processing* stage in which the consumer decompresses the downloaded dataset and then indexes it for local consumption. Every consumption task directly relies on efficient query resolution, thus, our second evaluation focuses on *query evaluation* performance.

Both post-processing and querying tasks require an RDF store enabling indexing and efficient SPARQL resolution. We choose three well-known stores for fair comparison with respect to HDT-FoQ:

- `RDF3X`[16] was recently reported as the fastest RDF store (Huang, Abadi, and Ren 2011).
- `Virtuoso`[17] is a popular store performing on relational infrastructure.
- `Hexastore`[18] is a well-known memory-resident store.

Post-processing. As stated, this task involves decompression and indexing in order to make queryable the compressed dataset retrieved from the publisher. Table 3 also organizes post-processing times for all datasets. It is worth noting that we compare our `HDT+lzma` against a baseline comprising `lzma` decompression and `RDF3X` indexing because it reports the best numbers. Cells containing ">24h" mean that the processes was not finished after 24 hours. Thus, indexing the `mashup` in our consumer is a very heavy task requiring a lot of computational resources and also wasting a lot of time.

HDT-based post-processing largely outperforms RDF3X for all original datasets in our setup. HDT performs decompression and indexing from ≈ 25 (`DBPedia`) to 114 (`Freebase`) times faster than RDF3X. This situation is due to two main reasons. On the one hand, HDT-encoded datasets are smaller than its counterparts in NTriples and it improves decompression

[16]RDF3X is available at http://www.mpi-inf.mpg.de/∼neumann/rdf3x/
[17]Virtuoso is available at http://www.openlinksw.com/
[18]Hexastore has been kindly provided by the authors.

Table 3: Overall client times (seconds). Baseline means that the file is downloaded in NTriples format, compressed using `lzma` and indexed using RDF-3X. HDT means that the file is downloaded in HDT, compressed with `lzma` and indexed using HDT-FoQ.

Dataset	Config.	Exchange	Decomp.	Index	Total
`LinkedMDB`	Baseline	9.61	5.11	111.08	125.80
	HDT	**6.25**	**1.05**	**1.91**	**9.21**
`DBLP`	Baseline	164.09	70.86	1387.29	1622.24
	HDT	**89.35**	**14.82**	**16.79**	**120.96**
`Geonames`	Baseline	174.46	87.51	2691.66	2953.63
	HDT	**118.29**	**19.91**	**44.98**	**183.18**
`DBPedia`	Baseline	1659.95	553.43	7904.73	10118.11
	HDT	**832.35**	**197.62**	**129.46**	**1159.43**
`Freebase`	Baseline	1910.86	681.12	58080.09	60672.07
	HDT	**891.90**	**227.47**	**286.25**	**1405.62**
`Mashup`	Baseline	3757.92	1238.36	>24h	>24h
	HDT	**1839.61**	**424.32**	**473.64**	**2737.57**

performance. On the other hand, HDT-FoQ generates its additional indexing structures (see Section 4.5.2) over the original HDT encoding whereas RDF3X first needs parsing the dataset and then building their specific indexes from scratch. Both features share an important fact: the most expensive processing was already done in the server side and HDT-encoded datasets are clearly better for machine consumption.

Exchange and post-processing times can be analyzed in together due to it is the total time than a consumer must wait until the data is able to be efficiently used in any application. Our integrated approach, around HDT encoding and data structures, completes all the tasks $8-43$ times faster than the traditional combination of compression and RDF indexing. It means, for instance, that the configured consumer retrieves and makes queryable `Freebase` in roughly 23 minutes using HDT, but it needs almost 17 hours to complete the same process over the baseline. In addition, we can see that indexing is clearly the heavier task in the baseline, whereas exchange is the longer task for us. However, in any case, we always complete exchange faster due to our achievements in space.

Querying. Once the consumer has made the downloaded data queryable, the infrastructure is ready to build on-top applications issuing SPARQL queries. The data volume emerges again as a key factor because it restricts the ways indexes and query optimizers are designed and managed.

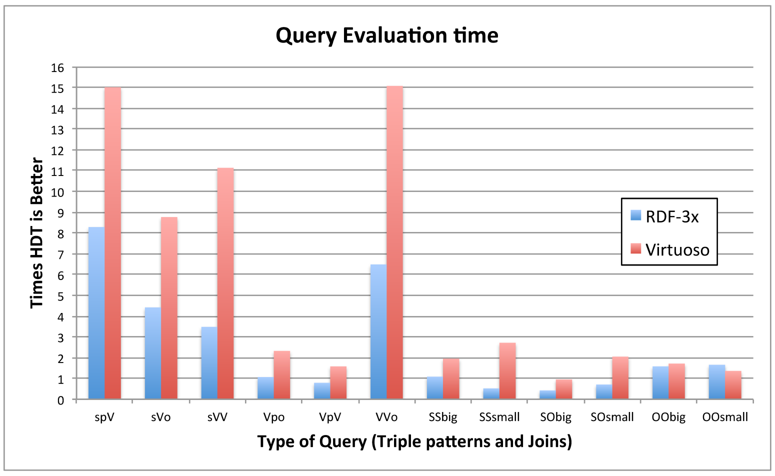

Figure 6: Comparison on querying performance on `Geonames`.

On the one hand, RDF3X and Virtuoso rely on disk-based indexes which are selectively loaded into main memory. Although both are efficiently tuned for this purpose, these I/O transfers result in very expensive operations that hinder the final querying performance. On the other hand, Hexastore and HDT-FoQ always hold their indexes in memory, avoiding these slow accesses to disk. Whereas HDT-FoQ enables all datasets in the setup to be managed in the consumer configuration, Hexastore is only able to index the smaller one, showing its scalability problems when managing Big Semantic Data.

We obtain two different sets of SPARQL queries to compare HDT-FoQ against the indexing solutions within the state-of-the-art. On the one hand, 5000 queries are randomly generated for each triple pattern. On the other hand, we also generate 350 queries of each type of two-way join, subdivided in two groups depending on whether they have a *small* or *big* amount of intermediate results. All these queries are run over `Geonames` in order to include both Virtuoso and RDF3X in the experiments. Note that, both classes of queries are resolved without the need of query planning, hence the results are clear evidence of how the different indexing techniques perform.

Figure 6 summarizes these querying experiments. The X-axis lists all different queries: the left subgroup lists the triple patterns, and the right ones represents all different join classes. The Y-axis means the number of times that HDT-FoQ is faster than its competitors. For instance, in the pattern `(S,V,V)` (equivalent to dereference the subject S), HDT-FoQ is more than 3 times faster than RDF3X and more than 11 times faster than Virtuoso. In general, HDT-FoQ always outperforms Virtuoso, whereas RDF3X is slightly faster for `(V,P,V)`, and some join classes. Nevertheless, we remain competitive in all theses cases and our join algorithms are still open for optimization.

4.7 Conclusions and Next Steps

This chapter presents basic foundations of Big Semantic Data management. First, we trace a route from the current data deluge, the concept of Big Data and the need of machine-processable semantics on the WWW. The Resource Description Framework (RDF) and the Web of (Linked) Data naturally emerge in this well-grounded scenario. The former, RDF, is the natural codification language for semantic data, combining the flexibility of semantic networks with a graph data structure that makes it an excellent choice for describing metadata at Web Scale. The latter, the Web of (Linked) Data, provides a set of rules to publish and link Big Semantic Data.

We justify the different and various management problems arising in Big Semantic Data by characterizing their main stakeholders by role (Creators/Publishers/Consumers) and nature (Automatic/Supervised/Human). Then, we define a common workflow Publication-Exchange-Consumption, existing in most applications in the Web of Data. The scalability problems arising to the current state-of-the-art management solutions within this scenario set the basis of our integrated proposal HDT, based on the W3C standard RDF.

HDT is designed as a binary RDF format to fulfill the requirements of portability (from and to other formats), compact ability, parsing efficiency (readiness for post-processing) and direct access to any piece of data in the dataset. We detail the design of HDT and we argue that HDT-encoded datasets can be directly consumed within the presented workflow. We show that lightweight indexes can be created once the different components are loaded into the memory hierarchy at the consumer, allowing for more complex operations such as joining basic SPARQL Triple Patterns. Finally, this compact infrastructure, called HDT-FoQ (HDT Focused on Querying) is evaluated toward a traditional combination of universal compression (for exchanging) and RDF indexing (for consumption).

Our experiments show how HDT excels at almost every stage of the publish-exchange-consumption workflow. The publisher spends a bit more time to encode the Big Semantic dataset, but in return, the consumer is able to retrieve it twice as fast, and the indexing time is largely reduced to just a few minutes for huge datasets. Therefore, the time since a machine or human client discovers the dataset until she is ready to start querying its content is reduced up to sixteen times by using HDT instead of the traditional approaches. Furthermore, the query performance is very competitive compared to state-of-the art RDF stores, thanks to the size reduction the machine can keep a vast amount of triples in main memory, avoiding slow I/O transferences.

There are several areas where HDT can be further exploited. We foresee a huge potential of HDT to support many aspects of the workflow Publish-Exchange-Consume. HDT-based technologies can emerge to provide supporting tools for both publishers and consumers. For instance a very useful tool for a publisher is setting up a SPARQL endpoint on top of an HDT file. As the experiments show, HDT-FoQ is very competitive on queries, but there is still plenty of room for SPARQL optimization, by leveraging efficient resolution of triple patterns, joins and query planning. Another useful tool for publishers is configuring a dereferenceable URI materialization from a given HDT. Here the experiments also show that performance will be very high because HDT-FoQ is really fast on queries with a fixed RDF subject.

Acknowledgments

This work was partially funded by MICINN (TIN2009-14009-C02-02); Science Foundation Ireland: Grant No.∼SFI/08/CE/I1380, Lion-II; Fondecyt 1110287 and Fondecyt 1-110066. The first author is granted by Erasmus Mundus, the Regional Government of Castilla y León (Spain) and the European Social Fund. The third author is granted by the University of Valladolid: programme of Mobility Grants for Researchers (2012)

References

Abadi, D., A. Marcus, S. Madden, and K. Hollenbach (2009). SW-Store: a vertically partitioned DBMS for Semantic Web data management. *The VLDB Journal 18*, 385–406.

Adida, B., I. Herman, M. Sporny, and M. Birbeck (Eds.) (2012). *RDFa 1.1 Primer*. W3C Working Group Note. http://www.w3.org/TR/xhtml-rdfa-primer/.

Akar, Z., T. G. Hala, E. E. Ekinci, and O. Dikenelli (2012). Querying the Web of Interlinked Datasets using VOID Descriptions. In *Proc. of the Linked Data on the Web Workshop (LDOW)*.

Alexander, K. (2008). RDF in JSON: A Specification for serialising RDF in JSON. In *Proc. of the 4th Workshop on Scripting for the Semantic Web (SFSW)*.

Alexander, K., R. Cyganiak, M. Hausenblas, and J. Zhao (2009). Describing Linked Datasets-On the Design and Usage of voiD, the 'Vocabulary of Interlinked Datasets'. In *Proc. of the Linked Data on the Web Workshop (LDOW)*.

Álvarez-García, S., N. Brisaboa, J. Fernández, and M. Martínez-Prieto (2011). Compressed k^2-triples for full-in-memory RDF engines. In *Proc. 17th Americas Conference on Information Systems (AMCIS)*, pp. paper 350.

Arenas, M., A. Bertails, E. Prud'hommeaux, and J. Sequeda (Eds.) (2012). *A Direct Mapping of Relational Data to RDF*. W3C Recommendation. http://www.w3.org/TR/rdb-direct-mapping/.

Arias, M., J. D. Fernández, and M. A. Martínez-Prieto (2011). An Empirical Study of Real-World SPARQL Queries. In *Proc. of 1st Workshop on Usage Analyss and the Web of Data (USEWOD)*. http://arxiv.org/abs/1103.5043.

Atemezing, G., O. Corcho, D. Garijo, J. Mora, M. Poveda-Villalón, P. Rozas, D. Vila-Suero, and B. Villazón-Terrazas (2012). Transforming Meteorological Data into Linked Data. *Semantic Web Journal [under review]*. http://www.semantic-web-journal.net/sites/default/files/swj281_0.pdf (accessed October 8, 2012).

Atre, M., V. Chaoji, M. Zaki, and J. Hendler (2010). Matrix "Bit" loaded: a scalable lightweight join query processor for RDF data. In *Proc. of the 19th World Wide Web Conference (WWW)*, pp. 41–50.

Auer, S., C. Bizer, G. Kobilarov, J. Lehmann, and Z. Ives (2007). Dbpedia: A nucleus for a web of open data. In *Proc. of the 6th International Semantic Web Conference (ISWC)*, pp. 11–15.

Baeza-Yates, R. and B. A. Ribeiro-Neto (2011). *Modern Information Retrieval - the concepts and technology behind search* (2 ed.). Pearson Education Ltd.

Beckett, D. (Ed.) (2004). *RDF/XML Syntax Specification (Revised)*. W3C Recommendation. http://www.w3.org/TR/rdf-syntax-grammar/.

Beckett, D. and T. Berners-Lee (2008). *Turtle - Terse RDF Triple Language*. W3C Team Submission. http://www.w3.org/TeamSubmission/turtle/.

Berners-Lee, T. (1998). *Notation3*. W3C Design Issues. http://www.w3.org/DesignIssues/Notation3.

Berners-Lee, T. (2002). Linked Open Data. What is the idea? http://www.thenationaldialogue.org/ideas/linked-open-data (accessed October 8, 2012).

Berners-Lee, T. (2006). Linked Data: Design Issues. http://www.w3.org/DesignIssues/LinkedData.html (accessed October 8, 2012).

Bizer, C., T. Heath, and T. Berners-Lee (2009). Linked Data - The Story So Far. *International Journal on Semantic Web and Information Systems 5*, 1–22.

Brickley, D. (2004). *RDF Vocabulary Description Language 1.0: RDF Schema*. W3C Recommendation. http://www.w3.org/TR/rdf-schema/.

Brisaboa, N., R. Cánovas, F. Claude, M. Martínez-Prieto, and G. Navarro (2011). Compressed String Dictionaries. In *Proc. of 10th International Symposium on Experimental Algorithms (SEA)*, pp. 136–147.

Cukier, K. (2010). Data, data everywhere. *The Economist* (February, 25). http://www.economist.com/opinion/displaystory.cfm?story_id=15557443 (accessed October 8, 2012).

Cyganiak, R., H. Stenzhorn, R. Delbru, S. Decker, and G. Tummarello (2008). Semantic sitemaps: Efficient and flexible access to datasets on the semantic web. In *Proc. of the 5th European Semantic Web Conference (ESWC)*, pp. 690–704.

De, S., T. Elsaleh, P. M. Barnaghi, and S. Meissner (2012). An Internet of Things Platform for Real-World and Digital Objects. *Scalable Computing: Practice and Experience 13*(1).

Dijcks, J.-P. (2012). Big Data for the Enterprise. *Oracle (white paper)* (January). http://www.oracle.com/us/products/database/big-data-for-enterprise-519135.pdf (accessed October 8, 2012).

Dumbill, E. (2012a). *Planning for Big Data*. O'Reilly Media.

Dumbill, E. (2012b). What is big data? *Strata* (January, 11). http://strata.oreilly.com/2012/01/what-is-big-data.html (accessed October 8, 2012).

Fernández, J. D., M. A. Martínez-Prieto, and C. Gutiérrez (2010). Compact Representation of Large RDF Data Sets for Publishing and Exchange. In *Proc. of the 9th International Semantic Web Conference (ISWC)*, pp. 193–208.

Fernández, J. D., M. A. Martínez-Prieto, C. Gutiérrez, and A. Polleres (2011). *Binary RDF Representation for Publication and Exchange (HDT)*. W3C Member Submission. http://www.w3.org/Submission/2011/03/.

Foulonneau, M. (2011). Smart semantic content for the future internet. In *Metadata and Semantic Research*, Volume 240 of *Communications in Computer and Information Science*, pp. 145–154. Springer Berlin Heidelberg.

García-Silva, A., O. Corcho, H. Alani, and A. Gómez-Pérez (2012). Review of the state of the art: discovering and associating semantics to tags in folksonomies. *The Knowledge Engineering Review 27*(01), 57–85.

González, R., S. Grabowski, V. Mäkinen, and G. Navarro (2005). Practical implementation of rank and select queries. In *Proc. of 4th International Workshop Experimental and Efficient Algorithms (WEA)*, pp. 27–38.

Grossi, R., A. Gupta, and J. Vitter (2003). High-order entropy-compressed text indexes. In *Proc. of 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 841–850.

Haas, K., P. Mika, P. Tarjan, and R. Blanco (2011). Enhanced results for web search. In *Proc. of the 34th International Conference on Research and Development in Information Retrieval (SIGIR)*, pp. 725–734.

Halfon, A. (2012). Handling big data variety. http://www.finextra.com/community/fullblog.aspx?blogid=6129 (accessed October 8, 2012).

Hausenblas, M. and M. Karnstedt (2010). Understanding Linked Open Data as a Web-Scale Database. In *Proc. of the 1st International Conference on Advances in Databases.*

Heath, T. and C. Bizer (2011). *Linked Data: Evolving the Web into a Global Data Space.* Synthesis Lectures on the Semantic Web: Theory and Technology, Morgan & Claypool.

Hey, T., S. Tansley, and K. M. Tolle (2009). Jim Gray on eScience: a transformed scientific method. In *The Fourth Paradigm.* Microsoft Research.

Hogan, A., A. Harth, J. Umbrich, S. Kinsella, A. Polleres, and S. Decker (2011). Searching and browsing linked data with swse: The semantic web search engine. *Journal of Web Semantics 9*(4), 365–401.

Hogan, A., J. Umbrich, A. Harth, R. Cyganiak, A. Polleres, and S. Decker (2012). An empirical survey of linked data conformance. *Web Semantics: Science, Services and Agents on the World Wide Web 14*(0), 14 – 44.

Huang, J., D. Abadi, and K. Ren (2011). Scalable SPARQL querying of large RDF graphs. *Proceedings of the VLDB Endowment 4*(11), 1123–1134.

Knoblock, C. A., P. Szekely, J. L. Ambite, S. Gupta, A. Goel, M. Muslea, K. Lerman, and P. Mallick (2012). Semi-Automatically Mapping Structured Sources into the Semantic Web. In *Proc. of the 9th Extended Semantic Web Conference (ESWC).*

Le-Phuoc, D., J. X. Parreira, V. Reynolds, and M. Hauswirth (2010). RDF On the Go: An RDF Storage and Query Processor for Mobile Devices. In *Proc. of the 9th International Semantic Web Conference (ISWC).* http://ceur-ws.org/Vol-658/paper503.pdf.

Lohr, S. (2012). The Age of Big Data. *The New York Times* (February, 11). http://www.nytimes.com/2012/02/12/sunday-review/big-datas-impact-in-the-world.html (accessed October 8, 2012).

Loukides, M. (2012). *What is Data Science?* O'Reilly Media.

Manola, F. and E. Miller (Eds.) (2004). *RDF Primer.* W3C Recommendation. www.w3.org/TR/rdf-primer/.

Martínez-Prieto, M., M. Arias, and J. Fernández (2012). Exchange and Consumption of Huge RDF Data. In *Proc. of the 9th Extended Semantic Web Conference (ESWC)*, pp. 437–452.

Martínez-Prieto, M., J. Fernández, and R. Cánovas (2012). Querying RDF Dictionaries in Compressed Space. *ACM SIGAPP Applied Computing Reviews 12*(2), 64–77.

Marz, N. and J. Warren (2013). *Big Data: Principles and best practices of scalable realtime data systems.* Manning Publications.

McGuinness, D. L. and F. van Harmelen (Eds.) (2004). *OWL Web Ontology Language Overview.* W3C Recommendation. http://www.w3.org/TR/owl-features/.

Neumann, T. and G. Weikum (2010). The RDF-3X Engine for Scalable Management of RDF data. *The VLDB Journal 19*(1), 91–113.

Prud'hommeaux, E. and A. Seaborne (Eds.) (2008). *SPARQL Query Language for RDF.* http://www.w3.org/TR/rdf-sparql-query/. W3C Recommendation.

Quesada, J. (2008). Human Similarity theories for the semantic web. In *Proceedings of the First International Workshop on Nature Inspired Reasoning for the Semantic Web.*

Ramakrishnan, R. and J. Gehrke (2000). *Database Management Systems.* Osborne/McGraw-Hill.

Schmidt, M., M. Meier, and G. Lausen (2010). Foundations of SPARQL Query Optimization. In *Proc. of the 13th International Conference on Database Theory (ICDT)*, pp. 4–33.

Schneider, J. and T. Kamiya (Eds.) (2011). *Efficient XML Interchange (EXI) Format 1.0.* W3C Recommendation. http://www.w3.org/TR/exi/.

Schwarte, A., P. Haase, K. Hose, R. Schenkel, and M. Schmidt (2011). FedX: optimization techniques for federated query processing on linked data. In *Proc. of the 10th International Conference on the Semantic Web (ISWC)*, pp. 601–616.

Selg, E. (2012). The next Big Step - Big Data. *GFT Technologies AG (technical report).* http://www.gft.com/etc/medialib/2009/downloads/techreports/2012.Par.0001.File.tmp/gft_techreport_big_data.pdf (accessed October 8, 2012).

Sidirourgos, L., R. Goncalves, M. Kersten, N. Nes, and S. Manegold (2008). Column-store Support for RDF Data Management: not All Swans are White. *Proc. of the VLDB Endowment 1*(2), 1553–1563.

Taheriyan, M., C. A. Knoblock, P. Szekely, and J. L. Ambite (2012). Rapidly integrating services into the linked data cloud. In *Proc. of the 11th International Semantic Web Conference (ISWC).*

Tran, T., G. Ladwig, and S. Rudolph (2012). Rdf data partitioning and query processing using structure indexes. *IEEE Transactions on Knowledge and Data Engineering 99*(PrePrints).

Tummarello, G., R. Cyganiak, M. Catasta, S. Danielczyk, R. Delbru, and S. Decker (2010). Sig.ma: Live views on the web of data. *Web Semantics: Science, Services and Agents on the World Wide Web 8*(4), 355–364.

Urbani, J., J. Maassen, and H. Bal (2010). Massive Semantic Web data compression with MapReduce. In *Proc. of the 19th International Symposium on High Performance Distributed Computing (HPDC) 2010*, pp. 795–802.

Volz, J., C. Bizer, M. Gaedke, and G. Kobilarov (2009). Discovering and Maintaining Links on the Web of Data. In *Proc. of the 9th International Semantic Web Conference (ISWC)*, pp. 650–665.

Weiss, C., P. Karras, and A. Bernstein (2008). Hexastore: Sextuple Indexing for Semantic Web Data Management. *Proc. of the VLDB Endowment 1*(1), 1008–1019.

Witten, I. H., A. Moffat, and T. C. Bell (1999). *Managing Gigabytes : Compressing and Indexing Documents and Images.* Morgan Kaufmann.