



Departamento de Inteligencia Artificial
Facultad de Informática

Federated Query Processing for the Semantic Web

A thesis submitted for the degree of
PhD Thesis

Author: Msc. Carlos Buil-Aranda
Advisor: Dr. Oscar Corcho García

January 2012

Tribunal nombrado por el Sr. Rector Magfco. de la Universidad Politécnica de Madrid,
el día.....de.....de 20....

Presidente :

Vocal :

Vocal :

Vocal :

Secretario :

Suplente :

Suplente :

Realizado el acto de defensa y lectura de la Tesis el día.....de.....de 20..... en la E.T.S.I.
/Facultad.....

Calificación

EL PRESIDENTE

LOS VOCALES

EL SECRETARIO

Abstract

The recent years have witnessed a constant growth in the amount of RDF data available on the Web. This growth is largely based on the increasing rate of data publication on the Web by different actors such governments, life science researchers or geographical institutes. RDF data generation is mainly done by converting already existing legacy data resources into RDF (e.g. converting data stored in relational databases into RDF), but also by creating that RDF data directly (e.g. sensors). These RDF data are normally exposed by means of Linked Data-enabled URIs and SPARQL endpoints.

Given the sustained growth that we are experiencing in the number of SPARQL endpoints available, the need to be able to send federated SPARQL queries across them has also grown. Tools for accessing sets of RDF data repositories are starting to appear, differing between them on the way in which they allow users to access these data (allowing users to specify directly what RDF data set they want to query, or making this process transparent to them). To overcome this heterogeneity in federated query processing solutions, the W3C SPARQL working group is defining a federation extension for SPARQL 1.1, which allows combining in a single query, graph patterns that can be evaluated in several endpoints.

In this PhD thesis, we describe the syntax of that SPARQL extension for providing access to distributed RDF data sets and formalise its semantics. We adapt existing techniques for distributed data access in relational databases in order to deal with SPARQL endpoints, which we have implemented in our federation query evaluation system (SPARQL-DQP). We describe the static optimisation techniques that we implemented in our system and we carry out a series of experiments that show that our optimisations significantly speed up the query evaluation process in presence of large query results and optional operators.

Resumen

En los últimos años se ha experimentado un crecimiento constante en los datos modelados en RDF. Este crecimiento está basado en la cada vez más frecuente publicación de datos en la Web. Estos datos son publicados por organizaciones de todo tipo, desde gobiernos (como el gobierno de los EEUU o del Reino Unido, con sus iniciativas data.gov y data.gov.uk respectivamente) a usuarios finales o redes de sensores distribuídas a lo largo del globo. Estos datos son normalmente creados en un formato diferente a RDF pero rápidamente traducidos a él, y además son expuestos en la Web mediante SPARQL endpoints y URLs enlazadas.

Dado este continuo crecimiento de datos en RDF que estamos experimentando, empiezan a aparecer herramientas para consultar dichos datos. Sin embargo, estas herramientas difieren en la forma de gestionar y presentar a los usuarios estas consultas. Para solucionar esta heterogeneidad de herramientas, el World Wide Web Consortium está definiendo una extensión al estándar de consultas SPARQL para permitir la federación de dichas consultas. Esta extensión permite combinar, en una sola consulta, patrones SPARQL que pueden ser evaluados en distintos almacenes de datos RDF.

En esta tesis doctoral se describe la sintaxis de la extensión para la federación de consultas SPARQL además de formalizar su semántica. También hemos analizado los problemas asociados a la distribución de datos en bases de datos relacionales y adaptadas algunas de las soluciones existentes en ese contexto a nuestro problema. También describimos cómo accedemos a fuentes de datos en RDF y cómo hemos implementado un sistema de evaluación de consultas para la extensión de federación de consultas de SPARQL, describiendo algunas optimizaciones estáticas que fueron identificadas durante el análisis formal del lenguaje. Finalmente, hemos llevado a cabo una serie de experimentos que prueban cómo estas optimizaciones disminuyen el tiempo de evaluación de las consultas de forma significativa.

To my family...

Acknowledgements

Es tiempo para los agradecimientos, y hay muchos que hacer, ya que ha habido mucha gente, que sin su apoyo, esta tesis nunca hubiese finalizado.

Primero me gustaría dedicarle esta tesis a mis padres, tanto a mi padre Carlos como a mi madre Concepción. Ellos han estado siempre ahí, desde el comienzo, en los malos momentos de mi carrera y en los buenos. Sin ellos, no sería como soy y no hubiese llegado hasta este punto. Así que mi agradecimiento eterno y la dedicación de esta tesis. También me gustaría dedicarle esta tesis a mi hermano Miguel y a mi abuela Concepción, por todo el amor que me han dado y me dan.

Quiero agradecer a mis amigos el haber estado también siempre ahí, apoyándome, con su fe infinita en mi criterio y en mi valía.

Sacar adelante una tesis es algo complejo, hay muchas decisiones que tomar, decidir qué investigar, qué camino seguir (a pesar de que a priori sabes por dónde van los tiros), y requiere de muchísimo esfuerzo ya que es un muy largo proceso. Sin la supervisión de Óscar, con todo lo que me ha enseñado, con todo lo que me ha facilitado las cosas, con sus rápidas revisiones y comentarios, con su apoyo, dudo que hubiese acabado esta tesis, y casi en plazo.

Por supuesto, en esta tesis han contribuido también otras muchas personas, Asun, la directora del grupo, Raúl, Miguel, otro Raúl, José, Boris, Víctor, etc. No puedo nombrarlos a todos porque se acaban las páginas. Ellos son solo unos pocos representantes del OEG que me han ayudado a finalizar esta tesis, así que gracias. También agradecer a la gente de iSOCO, en especial a José Manuel, ya que ahí fue donde realmente empezó mi trabajo como estudiante de doctorado.

Yo no entiendo la investigación sin colaborar con otros investigadores. Sin las estancias que hice, en Edimburgo y Santiago, esta tesis no sería la misma. En Edimburgo me encontré con un grupo de grandes profesionales, de los que aprendí mucho, y tengo mucho que agradecerles, en especial a Ally Hume, Bartek Dobrzelecki, Radek Ostrowski, Amy Krause,

Elias Theocharopoulos, Tilaye Alemu, Mike Jackson y Mario Antonioletti. Y en Chile, entre otras cosas, me encontré con una de las personas más inteligentes y apasionadas con su trabajo que he conocido, Marcelo Arenas, que me facilitó siempre las cosas, especialmente en este último año, que fue cuando más lo necesité. Parte de esta tesis también es suya.

Finalmente, también quiero agradecer y dedicar esta tesis a Leslie e Iker, mi nueva familia, y mi futuro.

Contents

List of Figures	ix
List of Tables	xi
1 Introduction	1
1.1 Thesis Structure	4
1.2 Dissemination Results	5
2 State of the Art	7
2.1 Representation and Query Languages for the Semantic Web	7
2.1.1 The Resource Description Framework (RDF)	7
2.1.2 RDF Schema	10
2.1.3 OWL	12
2.1.4 SPARQL	13
2.1.4.1 Basic Graph Patterns	15
2.1.4.2 Optional Matches	15
2.1.4.3 Alternative Matches	16
2.1.4.4 Query forms	16
2.1.4.5 The upcoming SPARQL 1.1	17
2.2 Setting the context: The Semantic Web and the Web of Linked Data	19
2.3 Data Integration and Query Federation	21
2.3.1 Classification of Data Integration Approaches	22
2.3.1.1 Materialised and Virtualised data integration	22
2.3.1.2 Procedural and Declarative Data Integration	23
2.3.2 Query Federation Systems	23
2.3.2.1 Query Processing in Query Federation Systems	26

CONTENTS

2.3.2.2	Query Plan Optimisation in Distributed Query Processing	28
2.3.2.3	Query Execution Optimisation in Distributed Query Processing . . .	30
2.3.2.4	Query Federation Architectures	32
2.3.3	Challenges and Limitations of Traditional Federated Query Processing Technol- ogy	34
2.3.4	Adaptive Query Operators	35
2.3.5	SPARQL optimisations	36
2.4	Ontology-Based Data Integration	38
2.4.1	Distributed SPARQL Query Processing Systems	38
2.4.1.1	Discussion on Query Federation System	40
2.4.2	RDB2RDF systems	43
2.5	Conclusions	44
3	Objectives and contributions	47
3.1	Objectives	47
3.1.1	Goals and Research Problems	47
3.2	Contributions to the State of the Art	49
3.3	Assumptions	50
3.4	Hypothesis	50
3.5	Restrictions	51
4	Problem formalisation	53
4.1	SPARQL Syntax	53
4.2	SPARQL Semantics	55
4.3	On Evaluating the SERVICE Operator in SPARQL queries	58
4.3.1	The notion of boundedness	58
4.3.2	The notion of service-safeness: Considering sub-patterns and nested SERVICE operators	62
4.4	Static Optimisations	63
4.4.1	Optimisation via well-designed patterns	64
5	Accessing Distributed RDF data stores	67
5.1	Introduction to Web service data access	67
5.2	Introduction to Distributed Query Processing	68
5.2.1	OGSA-DAI	69

5.2.1.1	OGSA-DAI workflows	71
5.2.1.2	OGSA-DAI Resources	71
5.2.1.3	OGSA-DAI Web services	72
5.2.2	OGSA-DQP	73
5.2.2.1	DQP Resource	73
5.2.2.2	Data Dictionary	73
5.2.2.3	SQL Expression Parser	74
5.2.2.4	Logical Query Plan (LQP) Builder	74
5.2.2.5	Query plan partitioning	74
5.2.2.6	Query plan to OGSA-DAI workflow translator	74
5.2.2.7	Workflow execution	75
5.2.2.8	OGSA-DAI activities	75
5.2.2.9	OGSA-DQP optimisers	76
5.2.3	OGSA-DAI Limitations	77
5.3	Accessing Distributed RDF data	78
5.3.1	Accessing RDF data sources	78
5.3.1.1	RDF Activities	80
5.3.2	Accessing RDB2RDF data resources	80
5.3.3	R2O resource	81
5.3.4	R2O Activities	82
5.3.4.1	Execution	83
5.4	SPARQL-DQP: Enabling Distributed SPARQL Query Processing in OGSA-DQP . . .	83
5.4.1	SPARQL-DQP Resource	84
5.4.2	RDF Data Dictionary	85
5.4.3	SPARQL Expression Parser	86
5.4.4	SPARQL Logical Query Plan Builder	86
5.4.5	Query plan partitioning	87
5.4.6	Query plan to OGSA-DAI workflow translator	87
5.4.7	Workflow execution	87
5.4.8	New OGSA-DAI activities	87
5.4.9	SPARQL-DQP optimisers	89
5.4.10	Federated SPARQL query execution	90
5.4.11	SPARQL-DQP Limitations	91

CONTENTS

5.4.12	SPARQL-DQP summarising	93
5.5	Conclusions	96
6	Evaluation	97
6.1	Benchmark Suite Design	97
6.2	Selection of Systems for the Evaluation	98
6.3	Evaluation 1	99
6.3.1	Data set description	100
6.3.2	Queries used in the evaluation	100
6.3.3	Results	101
6.4	Evaluation 2	102
6.4.1	Data sets description	103
6.4.2	Queries used in the evaluation	103
6.4.2.1	New queries used in the evaluation	108
6.4.3	Results	110
7	Conclusions and future work	115
7.1	Conclusions	115
7.2	Future work	117
A	Appendix A	119
A.1	Proofs	119
A.1.1	Proof of Theorem 1	119
A.1.2	Proof of Proposition 1	119
A.1.3	Proof of Theorem 2	122
A.1.4	Proof of Proposition 2	123
A.2	Evaluation 1 Queries	123
A.2.1	Bio2RDF Queries	123
A.3	Evaluation 2 Queries	125
A.3.1	Cross Domain Queries	125
A.3.2	Life Sciences Queries	128
A.3.3	SP2Bench Queries	130
	References	135

List of Figures

2.1	Graphical representation of an RDF triple	8
2.2	Graphical representation of several RDF triples	9
2.3	Graphical representation of several RDF triples with blank node	10
2.4	Linked Open Data cloud (as of September 2011)	20
2.5	Reference architecture for a federated Database Management System [SL90]	26
2.6	Most common phases of query processing	27
2.7	RDB2RDF systems working modes	44
4.1	Parse tree $\mathcal{T}(Q)$ for the graph pattern $Q = ((?Y, a, ?Z) \text{ UNION } ((?X, b, c) \text{ AND } (\text{SERVICE } ?X (?Y, a, ?Z))))$	62
5.1	Direct and indirect access modes in WS-DAI	68
5.2	Generic Architecture for SPARQL-DQP	69
5.3	Simple example of an OGSA-DAI workflow	70
5.4	Processing of an SQL Query in DQP	73
5.5	OGSA-DAI RDF extension	79
5.6	OGSA-DAI's RDB2RDF extension	81
5.7	OGSA-DAI-R2O extension (based on OGSA-DAI-RDF extension figures)	83
5.8	SPARQL-DQP Architecture	85
5.9	SPARQL 1.1 federated query execution	90
5.10	AST representing a SPARQL 1.1 federated query	91
5.11	Logical Query Plan representing a SPARQL 1.1 federated query	92

LIST OF FIGURES

List of Tables

2.1	Data Integration Level	40
2.2	Type of client-server System's Architectures	40
2.3	Query Planning Optimisation Techniques	40
2.4	SPARQL Specific Features	41
2.5	Query Execution Optimisations	42
2.6	Query or Data Transfer	43
2.7	Optimisation in client-server Architectures	43
2.8	Adaptive query optimisations	43
5.1	Data Integration Level in SPARQL-DQP	93
5.2	Type of SPARQL-DQP's client-server Architectures	93
5.3	SPARQL-DQP's Query Planning Optimisations	94
5.4	SPARQL Specific Features in SPARQL-DQP	94
5.5	Query Execution Optimisations in SPARQL-DQP	94
5.6	Query or Data Transfer in SPARQL-DQP	95
5.7	Optimisation in client-server Architectures in SPARQL-DQP	95
5.8	Adaptive query optimisations in SPARQL-DQP	96
5.9	Web-service based optimisations in SPARQL-DQP	96
6.1	Cross Domain Endpoint Characteristics	100
6.2	Evaluation 1 Query Results	102
6.3	Cross Domain Endpoint Characteristics	103
6.4	Life Sciences Endpoint Characteristics	103
6.5	Life Sciences Endpoint Characteristics	104
6.6	Results of Cross domain queries.	110

LIST OF TABLES

6.7	Results of life sciences domain queries.	111
6.8	Results of the SP2B evaluation.	112

1

Introduction

Recent years have witnessed a large and constant growth in the amount of data available on the Web. These data are made available in many different formats (as HTML tables, spreadsheets, inside PDF documents, as Web services, etc.). From these formats we focus on the Resource Description Framework (RDF) data model, a suite of recommendations proposed by the W3C, which are the basis for the Semantic Web and for the Web of Linked Data. RDF data are made available through the use of the HTTP protocol or exposed via SPARQL endpoints, which are RESTful¹ [Fie00] services that accept queries written in the SPARQL query language [PS08, HS10] over HTTP adhering to the SPARQL protocol [CFT08], as defined by the corresponding W3C SPARQL recommendation documents.

Several non-exhaustive, sometimes out-of-date or not continuously maintained, lists of SPARQL endpoints or data catalogues are available: CKAN², The Data Hub³, the W3C wiki⁴, etc. Many of these data sets are interlinked, as depicted graphically in the well-known Linked Open Data Cloud diagram⁵. This allows navigating through all of them and facilitates building complex queries by combining data from different, sometimes heterogeneous and normally physically distributed data sets. As of September 2011, the Linked Open Data Cloud diagram, which represents most of the linked data sets available on the Web, it lists more than 200 data sets, all of them containing links to other data sets and sharing common vocabularies. At the time of writing there are a total of 25,200,042,407 triples⁶ stored in these data sets, with 437,205,908 out links from them and 395,499,690 in links to them. This makes it possible to federate queries to several of these data sets, generating a rich and complete set of results.

¹Representational state transfer (REST)

²<http://ckan.org/>

³<http://thedatahub.org/>

⁴<http://www.w3.org/wiki/SparqlEndpoints>

⁵<http://lod-cloud.net>

⁶<http://www4.wiwiiss.fu-berlin.de/lodcloud/>

1. INTRODUCTION

However, the current SPARQL 1.0 [PS08] recommendation has an important limitation in terms of defining and executing queries that span across distributed data sets, since it only considers the possibility of executing these queries in isolation against a single SPARQL endpoint. For users willing to federate queries across a number of RDF data sets several systems have been created, implementing ad-hoc extensions of the query language and protocol or including additional information about data sources in the configuration of their SPARQL endpoint servers [QL08, SS08, SHH⁺11].

To homogenise this, the current SPARQL 1.1 working drafts [EP10] include a federation extension document (together with other extensions that are out of the scope for the purpose of this thesis). These extensions are now being under discussion with the aim of generating a new W3C recommendation in the coming months. The federation extension of SPARQL 1.1 includes two new operators in the query language: SERVICE and BINDINGS. The former (SERVICE) allows a SPARQL query endpoint to be specified inside a SPARQL query, to which a portion of the query will be delegated. This query endpoint may be known at the time the query is constructed, and hence the SERVICE operator can specify the IRI of the SPARQL endpoint where it will be executed; or it may be a variable that may be bound at query execution time after executing an initial SPARQL query fragment in one of the aforementioned RDF-enabled data catalogues (e.g. semantic CKAN¹), so that potential SPARQL endpoints that can answer the rest of the query can be obtained and used. The latter (BINDINGS) allows results to be transferred that are used to constrain a query: these will normally come from previous executions of other queries or from constraints specified in user interfaces that then transform these constraints into SPARQL queries.

Not only are the syntax and semantics of the query language important for distributed query processing, but optimisation is also of key importance in this context. Querying a set of different SPARQL endpoints requires the generation of an optimal query plan to deliver as soon as possible results to the end user. Many problems may appear when querying distributed data sets, and the system should be able to act accordingly. Problems with network latency or server availability may appear. Besides, the amount of data returned by the remote data set may vary depending on the server. Normally SPARQL endpoints limit the amount of data returned for each query ranging from 1000 to 5000 results by default. For accessing these amounts of data, query plan optimisations might not be very necessary (the time required to transfer these little amounts of data should not be high). But if these limitations are removed, and in one single query hundreds of thousands of results are returned, its processing and transfer across the network will be very costly.

¹<http://semantic.ckan.net/>

For example, imagine a query to the Bio2RDF data set, which is divided in 42 SPARQL endpoints. Some of these endpoints are Pubmed¹ (797,000,000 triples) which contains 15 million citations for biomedical articles from the MEDLINE² bibliographic resource and other life science journals, Taxonomy³ (4,759,509 triples) which is the database of the International Sequence Database Collaboration containing the names of all organisms that are represented in the sequence databases with at least one nucleotide or protein sequence, UniProt⁴ (338,602,962 triples), storing the Universal Protein Resource which provides a data resource with information about protein sequences or KEGG⁵ (84,715,161 triples) which is a collection of pathway maps representing molecular interactions for cellular processes or human diseases. In such query scenario SPARQL endpoints will time out due to the data will need time to be transferred between nodes and servers and to be processed. Besides, endpoints may unpredictably become blocked, suddenly stopping producing results. Thus, it is necessary for the query execution engine to take into account all these problems when executing queries against a set of SPARQL endpoints, and produce results as quickly as data arrives.

In this thesis we formalise the access to distributed RDF data sets by proposing the syntax and semantics of these federation extensions for SPARQL 1.1. We define the constraints that have to be considered in their use (which is currently not too restrictive) in order to be able to provide pragmatic implementations of query evaluators. For instance, if we use a variable when specifying the SPARQL endpoints, we may imagine that a naïve implementation may need to go through all existing SPARQL endpoints on the Web evaluating that query fragment before being able to provide a result, something that can be considered unfeasible in practical terms. For our purpose, we define the notions of service-boundedness and service-safeness, which ensure that the SERVICE operator can be safely evaluated.

In addition, we implement the static optimisations proposed in [PAG09], using the notion of well-designed patterns. These optimisations prove to be effective in the optimisation of queries that contain the OPTIONAL operator, which is demonstrated to be the most costly operator in SPARQL [PAG09, MSL10]. This has also important implications in the number of tuples being transferred and joined in federated queries, and hence our implementation benefits from this. Other complementary works implement adaptive query processing techniques [AVL⁺11, LKMT11] which follow a dynamic approach for optimisation, by adapting query plans to the specific conditions of the query/execution environment.

As a result of our work, we have not only formalised these notions, but we have also implemented a system that supports the current SPARQL 1.1 federation extension specification and makes use of

¹<http://pubmed.bio2rdf.org/sparql>

²<http://www.ncbi.nlm.nih.gov/pubmed/>

³<http://taxonomy.bio2rdf.org/sparql>

⁴<http://uniprot.bio2rdf.org/sparql>

⁵<http://kegg.bio2rdf.org/sparql>

1. INTRODUCTION

these optimisations. This system, SPARQL-DQP (which stands for SPARQL Distributed Query Processing), is built on top of the OGSA-DAI (an extensible framework for accessing distributed and heterogeneous data via the use of data workflows) and OGSA-DQP (the Distributed Query Processing extension of OGSA-DAI, which optimises the access to distributed OGSA-DAI data resources) infrastructures [Ant07, DT10], which can deal with large amounts of data in distributed settings by supporting an indirect access pattern.

In this thesis we provide a novel implementation of an RDF data access mechanism that follows the OGF WS-DAI-RDF specification [PKL11], adding a Web service layer between the data and its users. We distinguish between two types of RDF data repositories: remote RDF data repositories, which are normally exposed by SPARQL endpoints (or by direct RDF database connections) and RDF data wrappers that provide RDF-based access to resources like text files, XML files, thesauri, relational databases, etc. In this thesis we focus in accessing both types of data resources and for the latter we provide an implementation of a relational database wrapper.

1.1 Thesis Structure

The remainder of the thesis is organised as follows:

- In Chapter 2 we analyse the current state of the art of the topics discussed in this work, namely the Semantic Web and its core languages and technologies, using query federation systems and ontology-based data integration systems. We also identify the current limitations of these approaches, what leads to the work that we present here.
- Chapter 3 describes the objectives and main contributions of this thesis, based on the previous limitations. We also describe the assumptions, hypotheses and restrictions of our work.
- Chapter 4 provides the theoretical formalisation of the problem. In this section we first provide a syntax definition of the federated extensions of the query language, and then we describe its semantics. Finally we propose static optimisation techniques that can be applied in the evaluation of these queries.
- Chapter 5 describes the systems that we have implemented which demonstrate the hypothesis described in Chapter 3. In this chapter we first describe how we access to a single remote RDF data source, and then we describe how to access a set of distributed RDF data sources. We also describe the optimisation techniques used for speeding up the execution of federated SPARQL queries.

- Chapter 6 provides an evaluation of the systems described in the previous section and compares them with the state of the art systems. We show the benefits of the optimisations proposed in Chapter 4 and implemented in Chapter 5.
- Chapter 7 describes the main conclusions of this thesis, and the proposed future lines of research in this area.

1.2 Dissemination Results

Our work has been presented in the following international workshops, conferences and standardisation activities:

- Our contribution in Chapter 4 has been partially included in: Eric Prud'hommeaux, Carlos Buil-Aranda, SPARQL 1.1 Federated Query, W3C Last Call document, and published in: Carlos Buil-Aranda, Marcelo Arenas and Oscar Corcho: Semantics and optimisation of the SPARQL 1.1 federation extension, Proceedings of the 8Th Extended Semantic Web Conference (ESWC2011). This paper received the **Best Research Paper Award**.
- Part of our contribution to Chapter 5 has been included in:: Steven J. Lynden, Oscar Corcho, Isao Kojima, Mario Antonioletti, Carlos Buil Aranda: Open Standards for Service-Based Database Access and Integration. Grid and Cloud Database Management 2011: 3-21
- Chapter 5 has been published in: Carlos Buil-Aranda, Oscar Corcho, Amy Krause., Robust service-based semantic querying to distributed heterogeneous databases, 8Th International Workshop on Web Semantics - WebS '09 [BACK09].
- A complete summary of this thesis has been submitted to the Journal of Web Semantics. Carlos Buil-Aranda, Marcelo Arenas, Oscar Corcho and Axel Polleres: Federating Queries in SPARQL1.1. Query: Syntax, Semantics and Evaluation. We are waiting for the corresponding reviews.

1. INTRODUCTION

2

State of the Art

In this chapter we present the current state of the art in querying distributed RDF data repositories. Besides we provide an analysis of existing approaches, techniques, tools and systems, for accessing distributed RDF and non-RDF data highlighting their main characteristics. Finally, we review the existing work for querying distributed RDF data sources and in integrating them. We analyse the methods and techniques used in these approaches, some of which are applied in our federated SPARQL query evaluation system.

2.1 Representation and Query Languages for the Semantic Web

This section provides an overview of the core Semantic Web concepts, that form the basis of our work, including the Resource Description Framework (RDF), RDF Schema, the Web Ontology Language (OWL), and the SPARQL Protocol and RDF Query Language. The contents of this section are based on the work presented in [Lan10] and reuse examples from there and from the corresponding W3C specifications.

2.1.1 The Resource Description Framework (RDF)

The Resource Description Framework [GK04] is the basic data model used in the Semantic Web. RDF extends the linking structure of the Web by using URIs [TBL05] to name the relationships between things as well as naming the two ends of the link (this is usually referred to as a "triple", represented in Figure 2.1). Using this simple model, allows structured and semi-structured data to be mixed, exposed, and shared across different applications. Any information added by higher-level components

2. STATE OF THE ART



Figure 2.1: Graphical representation of an RDF triple

like RDF Schema and OWL is modelled within RDF. The current RDF specification is split into six W3C Recommendations:

- The RDF Primer [FM04] describes the basic concepts of RDF. The RDF Primer also describes how to define vocabularies using the RDF Vocabulary Description Language (also called RDF Schema).
- RDF Concepts and Abstract Syntax [GK04] defines an abstract syntax on which RDF is based, and which serves to link its concrete syntax to its formal semantics. It also includes discussion of design goals, key concepts, data typing, character normalisation and handling of URI references.
- RDF/XML Syntax Specification [Bec04] defines an XML syntax for RDF in terms of Namespaces in XML, the XML Information Set and XML Base.
- RDF Semantics [Hay04] describes the semantics and corresponding complete systems of inference rules for RDF and RDF(S).
- RDF Vocabulary Description Language (RDF Schema) [DB04] defines a precise semantics, and corresponding complete systems of inference rules, for the Resource Description Framework (RDF) and RDF Schema (RDFS).
- RDF Test Cases [JG04] describes the RDF Test Cases deliverable for the RDF Core Working Group as defined in the Working Group's Charter.

The main concept of an RDF graph is a triple, which is represented as $\langle s, p, o \rangle$, denoting *subject*, *predicate* and *object*. The triple $\langle \text{http} : // \text{example.org/me}, \text{foaf} : \text{name}, \text{"Bob"} \rangle$ means that the resource *http://example.org/me* has as name *Bob*. In this example (represented in Figure 2.1), the subject is the resource with the URI *http://example.org/me*, the predicate is *foaf:name*, and the literal *Bob* is the object. The next lines depict a more complex RDF graph in Turtle syntax¹:

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix ex: <http://example.org/> .

ex:me foaf:homepage <http://www.example.org/> .
ex:me foaf:name "Bob" .
```

¹<http://www.w3.org/TeamSubmission/turtle/>

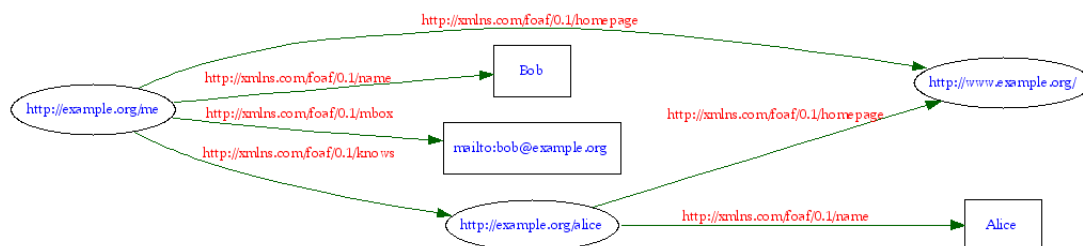


Figure 2.2: Graphical representation of several RDF triples

```
ex:me foaf:mbox "mailto:bob@example.org" .
ex:me foaf:knows ex:alice .
ex:alice foaf:name "Alice" .
ex:alice foaf:homepage <http://www.example.org/> .
```

The information contained in the graph is the conjunction of all six sentences or statements. Only resources can be used as subjects in RDF statements. The URI *http://example.org/me* is an RDF resource representing the person with the predicates *foaf:name*, *foaf:homepage* and *foaf:mbox* with values *Bob*, *http://www.example.org/* and *mailto:bob@example.org* respectively. In these data the relation *ex:me foaf:knows ex:alice* is also represented, meaning that the resource *ex:me* knows the resource *ex:alice*, which also has its own properties. Figure 2.2 represents such graph.

The *rdf:type* predicate is used to classify resources in RDF:

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix ex: <http://example.org/> .

ex:me rdf:type foaf:Person
```

This example states that *ex:me* is a person.

Nodes that represent a resource in an RDF graph but are not given a URI or a literal are called blank nodes. This resource has no URI associated, and represents an anonymous resource that can only be used as the subject or object in an RDF triple. Figure 2.3 demonstrates the usage of a blank node. Blank nodes in an RDF graph serialisation are limited to the scope of that serialisation for that particular RDF graph, i.e. the node *p1* in a graph G_1 example does not represent the same node as a node named *p1* in graph G_2 .

RDF literals can be either plain literals (simple strings), or typed literals. Typed literals have a data type tag for representing its type. For example, in Turtle syntax the typed literal `"12.34"^^xsd:float` represents the real number 12.34. These tags are compatible with the data type abstraction used in XML Schema and it is also possible to define custom data types. Plain literals can additionally have a

2. STATE OF THE ART

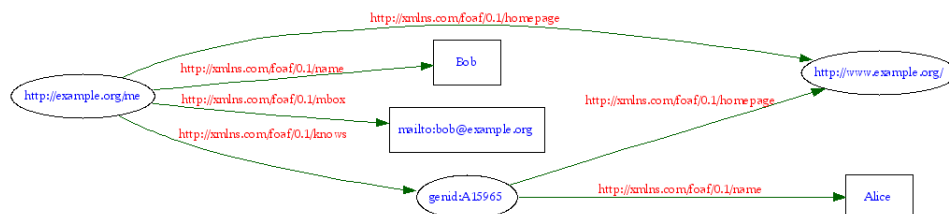


Figure 2.3: Graphical representation of several RDF triples with blank node

language tag stating the corresponding language used: for instance, the literal "This is a text written in English"@en indicates that the language of the plain text literal is English.

RDF can be serialised in four formats, even though, two of these formats are subsets of one of the others. The first serialisation format is RDF/XML [Bec04] which provides a XML serialisation for RDF data. This is the serialisation format recommended in the RDF specifications. The W3C also provides the Notation 3¹ format (or N3) which is a serialisation format easier to read for humans. N-Triples² and Turtle³ are subsets of N3, allowing to describe the same RDF triples in a simpler way.

Next, we provide introduce first the formal notions about RDF (taken mainly from [PAG09]). This notions can also be found in 4. Assume there are pairwise disjoint infinite sets I , B , and L (IRIs [DS05], Blank nodes, and Literals, respectively). Then a triple $(s, p, o) \in (I \cup B) \times I \times (I \cup B \cup L)$ is called an *RDF triple*, where s is called the *subject*, p the *predicate* and o the *object*. An *RDF graph* is a set of RDF triples.

Moreover, assume the existence of an infinite set V of variables disjoint from the above sets, and leave UNBOUND to be a reserved symbol that does not belong to any of the previously mentioned sets.

2.1.2 RDF Schema

RDF Schema⁴ extends the RDF Core vocabulary. RDF(S) contains several predefined concepts like `rdfs:Class`, `rdfs:Property`, etc., which are used to define custom classes and properties.

```
@prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs:  <http://www.w3.org/2000/01/rdf-schema#> .
@prefix ex:    <http://example.org/> .

ex:bob    rdf:type        ex:Person .
ex:alice  rdf:type        ex:Person .
ex:Person rdfs:subClassOf ex:animal .
ex:works  rdfs:range      ex:animal .
```

¹<http://www.w3.org/DesignIssues/Notation3>

²<http://www.w3.org/TR/rdf-testcases/>

³<http://www.w3.org/TeamSubmission/turtle/>

⁴<http://www.w3.org/TR/rdf-schema/>

The RDF(S) class list contains:

- `rdfs:Resource` which is the class of everything. All things described by RDF are resources.
- `rdfs:Class` declares a resource as a class for other resources.
- `rdfs:Literal` takes literal values such as strings and integers. Property values such as textual strings are examples of RDF literals. Literals may be plain or typed.
- `rdfs:Datatype` is the class of data types. `rdfs:Datatype` is both an instance of and a subclass of `rdfs:Class`. Each instance of `rdfs:Datatype` is a subclass of `rdfs:Literal`.
- `rdf:XMLLiteral` is the class of XML literal values. `rdf:XMLLiteral` is an instance of `rdfs:Datatype` (and thus a subclass of `rdfs:Literal`).
- `rdf:Property` is the class of properties.

The RDF(S) property list contains:

- `rdfs:domain` of an `rdf:property` declares the class of the subject in a triple whose second component is the predicate.
- `rdfs:range` of an `rdf:property` declares the class or data type of the object in a triple whose second component is the predicate.
- `rdfs:subClassOf` allows one to declare hierarchies of classes.
- `rdfs:subPropertyOf` is an instance of `rdf:Property` that is used to state that all resources related by one property are also related by another.
- `rdfs:label` is an instance of `rdf:Property` that may be used to provide a human-readable version of a resource's name.
- `rdfs:comment` is an instance of `rdf:Property` that may be used to provide a human-readable description of a resource.

Other properties:

- `rdfs:seeAlso` is an instance of `rdf:Property` that is used to indicate a resource that might provide additional information about the subject resource.

2. STATE OF THE ART

- `rdfs:isDefinedBy` is an instance of `rdf:Property` that is used to indicate a resource defining the subject resource. This property may be used to indicate an RDF vocabulary in which a resource is described.

In summary, RDF(S) provides a vocabulary that allows us to deal with inheritance of classes and properties or typing, among other features, providing the basic elements for defining more complex relationships between the elements of the data represented in the RDF model.

2.1.3 OWL

The Web Ontology Language (OWL)¹ [DS04] is a knowledge representation language, based on the description logic formalism [BCM⁺10]. OWL provides greater expressivity for describing domain knowledge than that supported by RDF, and RDF Schema. OWL provides additional vocabulary as well as formal semantics. For instance, it allows the cardinalities of properties to be expressed, universal and existential restrictions on properties and classes, algebraic characteristics of properties, etc.

OWL assumes the Open World Assumption [RNC⁺96]: this concept assumes that the given knowledge in a knowledge base should always be considered as incomplete and everything asserted into it is true unless the contrary is stated, in contrast to the closed world assumption which states that everything that is not in the knowledge base does not exist or is false. An example of this feature is the following: `< ex : me, rdf : type, foaf : Person >` states that I am a person, but it does not explicitly say that I am not a student or an engineer. Thus, when querying the knowledge base for all engineers or students, the resource `ex : me` should be among the results. In contrast when querying for the same resources in a relational database, no results should appear.

OWL does not include in its features the unique name assumption [RNC⁺96]. The unique world assumption states that different identifiers must refer to different entities in the real world. That is, it can not be said that `ex : me` is the same entity than `< http : //example.org/bob >`. Special predicates for asserting equivalence of resources exist and should be used in that case.

OWL also provides different flavours of the language. In OWL 1 the following variants are available: OWL Lite (lowest complexity providing a few more features than RDF(S)), OWL DL (reasoning over an OWL DL ontology may terminate in a finite time) and OWL full (maximum expressiveness but with no termination guarantee), depending on the expressivity of the language and the axioms used

¹<http://www.w3.org/TR/owl-features/>

in the ontology. In OWL 2¹ [HKP⁺09] other language profiles² are provided with different levels of expressivity:

OWL EL This profile captures the expressive power used by many such ontologies and is a subset of OWL 2 for which the basic reasoning problems can be performed in polynomial time with respect to the size of the ontology.

OWL 2 QL In this profile, conjunctive query answering can be implemented using conventional relational database systems. Using a suitable reasoning technique, sound and complete conjunctive query answering can be performed in LOGSPACE with respect to the size of the data (assertions).

OWL 2 RL This version is designed to accommodate OWL 2 applications that can trade the full expressivity of the language for efficiency, as well as RDF(S) applications that need some added expressivity. OWL 2 RL reasoning systems can be implemented using rule-based reasoning engines. The ontology consistency, class expression satisfiability, class expression subsumption, instance checking, and conjunctive query answering problems can be solved in polynomial time with respect to the size of the ontology.

The main differences between OWL 1 and 2 are the incorporation of the new flavours in the language, in which all three profiles are more restrictive than OWL DL. OWL 2 also adds new features which can be summarised as syntactic sugar to make some common statements easier to say (i.e. DisjointUnion, DisjointClasses, NegativeObjectPropertyAssertion and NegativeDataPropertyAssertion), new constructs that increase expressivity (i.e. self restriction, property qualified cardinality restrictions, reflexive, irreflexive, and asymmetric object properties, disjoint properties. etc.), extended support for data types (i.e. Extra data types and data type Restrictions or N-ary data types among others), simple metamodeling capabilities or extended annotation capabilities. In [GW09] there is described in more detail all these new features.

2.1.4 SPARQL

In this section we informally describe the SPARQL query language. For readers interested in a more formal definition, we refer them to Chapter 4. SPARQL is a graph pattern matching language for RDF. SPARQL is an acronym for either: SPARQL Protocol and RDF Query Language or Simple Protocol and RDF Query Language. Both definitions were coined in the Data Access Working Group (being

¹<http://www.w3.org/TR/owl2-overview/>

²<http://www.w3.org/TR/owl-profiles/>

2. STATE OF THE ART

preferred the former), formerly known as the SPARQL-WG. For instance, given the RDF graph depicted in Figure 2.2, we might be interested in obtaining the people that Bob knows. The following triple pattern represents such query:

```
<http://example.org/me> foaf:knows ?person
```

In the above SPARQL query we select all those triples from the source graph that have the subject *<http://example.org/me>* and the predicate *foaf:knows*. The object is represented as a free variable which can be instantiated with any valid value entailed in the graph. Variables in SPARQL are represented using a question mark before the variable name (in the example *?person* represents a variable). The syntax for triples is the same syntax that Turtle and Notation 3 use for representing triples.

The variable *?person* is bound to three different resources which are the objects of the remaining triples. The result of a SPARQL SELECT query is a multiset that can be represented as a table, as we have done with the previous examples. SPARQL results are represented with a table with all the query variables in the column headers and each solution mapping is represented in the following table rows. The query results for the above example query can be represented as shown in the Table below. As we have seen in the previous tables, the solution's mappings form an unordered multiset, thus the order of that solution mappings is irrelevant. Empty rows mean that the corresponding variables are unbound. The solutions form a multiset or bag containing three solution mappings from the query variables (here we only have *?person*) to their bound values

?person
<http://example.org/alice>
<http://example.org/alan>
<http://example.org/carol>

The triple pattern restricts the graph to a subset of triples that match the pattern. The least restrictive triple pattern possible is *?s ?p ?o*. It selects the entire graph and can be used to retrieve all the information entailed.

The results for a query matching all the triple patterns of the graph represented in Figure 2.2 are the following:

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix ex: <http://example.org/> .

ex:me foaf:name "Bob" .
ex:me foaf:knows ex:alice .
ex:me foaf:knows ex:alan .
```


2.1 Representation and Query Languages for the Semantic Web

```
ex:me foaf:knows ex:carol .  
ex:alice foaf:name "Alice" .  
ex:alice foaf:knows ex:alan .  
ex:alan foaf:name "Alan" .
```

These results are represented as a SPARQL query results in the following table:

?s	?p	?o
<http://example.org/me>	<http://xmlns.com/foaf/0.1/name>	"Bob"
<http://example.org/me>	<http://xmlns.com/foaf/0.1/knows>	<http://example.org/alice>
<http://example.org/me>	<http://xmlns.com/foaf/0.1/knows>	<http://example.org/alan>
<http://example.org/me>	<http://xmlns.com/foaf/0.1/knows>	<http://example.org/carol>
<http://example.org/alice>	<http://xmlns.com/foaf/0.1/name>	"Alice"
<http://example.org/alice>	<http://xmlns.com/foaf/0.1/knows>	<http://example.org/alan>
<http://example.org/alan>	<http://xmlns.com/foaf/0.1/name>	"Alan"

2.1.4.1 Basic Graph Patterns

In the previous example we only obtained the URI of resources, which are not very useful for persons. If we want to obtain the names of those RDF resources it is necessary to add another triple pattern that selects the *foaf:name* property for each *?person*. This can be achieved by the following basic graph pattern (BGP), which consists of two triple patterns:

```
{ <http://example.org/me> foaf:knows ?person .  
  ?person foaf:name ?name . }
```

Note that a BGP is a block delimited with braces. Similar to a set of RDF triples, the BGP can be represented as a graph

?person	?name
<http://example.org/alice>	"Alice"
<http://example.org/alan>	"Alan"

2.1.4.2 Optional Matches

In the previous example, there are only two solutions, because one of the persons in the source graph does not have an associated *foaf:name*. If the result can optionally include the person name, but still contain the person URI, the OPTIONAL keyword can be used:

```
{ <http://example.org/me> foaf:knows ?person .  
  OPTIONAL { ?person foaf:name ?name . } }
```

2. STATE OF THE ART

The solution mappings produced by the first BGP are merged with the solution mappings produced by the second BGP based on optional semantics, which means that if two bindings are not compatible, the left bindings are still streamed through as results:

?person	?name
<http://example.org/alice>	"Alice"
<http://example.org/alan>	"Alan"
<http://example.org/carol>	

2.1.4.3 Alternative Matches

The SPARQL union is the set theoretical union of two results sets. Therefore, the SPARQL union is different from the SQL union. In SQL the schemes (i.e. columns) of both sides of the union have to be compatible. This is not obligatory in SPARQL. When combining BGPs with the union operation, both BGPs can share (or not) common variables. They can even have distinct sets of variables. The union in SPARQL is the union of the solution mappings from the first and the second BGP. The following example shows the SPARQL union made against the previous graph:

```
{ { <http://example.org/me> foaf:knows ?person . }  
  UNION { ?person foaf:name ?name . } }
```

The result of this query is the union of both BGP results. The first BGP selects persons known by *http://example.org/me* including their names and the second BGP selects all persons with names.

?person	?name
<http://example.org/alice>	
<http://example.org/alan>	
<http://example.org/carol>	
<http://example.org/alice>	"Alice"
<http://example.org/alan>	"Alan"
<http://example.org/carol>	
<http://example.org/me>	"Bob"

2.1.4.4 Query forms

Up until now we have only seen one type of query in action: SELECT. However, this is not the only one we can use. In SPARQL there are several query forms that allow several query types to be generated, all of them are based on graph pattern matching. A SELECT query is used to select elements from the data. A DESCRIBE query returns information about the resources that match a graph pattern in the form of

another RDF graph. A CONSTRUCT query returns a graph based on the solutions constructed from the graph pattern indicated in the query. Finally ASK queries return true or false if there are any possible solutions to the query.

2.1.4.5 The upcoming SPARQL 1.1

The next version of SPARQL, SPARQL 1.1 is being developed by the W3C SPARQL-WG¹. This new version is an extension to SPARQL 1 for adding elements that could not be included previously. It extends the SPARQL 1.0 query document with subqueries and aggregation operators and adds several new documents with new functionality: SPARQL 1.1 Update, Entailment Regimes, Service Description, Federated Query, Protocol, Property Paths and JSON, CSV and TSV query result formats:

SPARQL 1.1 Query² This document is the extension of the main query document of SPARQL 1.0.

The new features include aggregation functions that allow operations such as counting, numerical min/max/average and so on, by operating over columns of results. Subqueries are also specified in this document, providing a way to nest the results of a query within another query. For instance, a user can identify the 10th latest blog posts created in a web log, with a single author name for each blog post³. The negation feature, implicitly present in SPARQL 1.0 (via the filtering of unbound variables) is also present in the new query document with the NOT EXISTS filter and the MINUS keyword. NOT EXISTS deals with negation testing, i.e. whether a pattern exists in the data, given the bindings already determined by the query pattern. The NOT EXISTS filter removes matches based on the evaluation of two patterns. Project expressions: Projecting expressions represents the ability for SPARQL SELECT queries to project any SPARQL expression, rather than only variables. A projected expression might be a variable, a constant URI, a constant literal, or an arbitrary expression (including function calls) on variables and constants.

SPARQL 1.1 Update⁴ This SPARQL extension provides features for allowing the insertion of new RDF triples into an RDF data set exposed by an SPARQL endpoint. This provides the ability for inserting triples into an RDF graph, deleting triples from an RDF graph, loading an RDF graph, clearing an RDF graph, creating a new RDF graph all in the Graph Store specified by the endpoint address. It is also possible to drop an RDF graph from the Graph Store and copy, move, or add the content of one RDF graph in a Graph Store to another. It also allows a group of update operations to be performed as a single action.

¹<http://www.w3.org/2009/sparql/wiki/>

²<http://www.w3.org/2009/sparql/docs/query-1.1/>

³<http://www.w3.org/TR/2009/WD-sparql-features-20090702/>

⁴<http://www.w3.org/2009/sparql/docs/update-1.1/>

2. STATE OF THE ART

SPARQL 1.1 Protocol¹ describes the SPARQL Protocol, a means of conveying SPARQL queries and updates from clients to query processors. The SPARQL Protocol is described in two ways: first, as an abstract interface independent of any concrete realisation, implementation, or binding to another protocol; second, as an HTTP binding of this interface.

SPARQL 1.1 Service Description² provides a method for discovering and a vocabulary for describing SPARQL services made available via the SPARQL 1.1 Protocol for RDF. These descriptions provide a mechanism by which a client or end user can discover information about the SPARQL service such as supported extension functions and details about the available data set.

SPARQL 1.1 Entailment Regimes³ provides basic entailment regime for SPARQL. Now SPARQL endpoints may be able to support a certain form of entailment such as RDFS. Users, when querying the remote endpoint, will get in their results also those that reflect all RDFS consequences. Is left to the SPARQL endpoint what entailment type is implementing.

SPARQL 1.1 Graph Store HTTP Protocol⁴ describes the protocol for updating and fetching RDF graph content from a Graph Store over HTTP in the REST style.

SPARQL 1.1 Federated Query⁵ defines the syntax and semantics of SPARQL 1.1 Federated Query extension for executing queries distributed over different SPARQL endpoints. The SERVICE keyword extends SPARQL 1.1 to support queries that merge data distributed across the Web. This is the document we are interested in the context of this PhD thesis and to which we have contributed our results.

SPARQL 1.1 Property Paths⁶ In this document property paths matching is SPARQL queries is defined. A property path is a possible route through a graph between two graph nodes. A trivial case is a property path of length exactly 1, which is a triple pattern. This document is included in the main SPARQL 1.1 query document.

SPARQL 1.1 TSV, CSV results⁷ describes the use of the Tab Separated Values and Comma Separated Values formats in the SPARQL results from SELECT queries.

SPARQL 1.1 JSON results⁸ describes the representation of SELECT and ASK query results using

¹<http://www.w3.org/2009/sparql/docs/protocol-1.1/>

²<http://www.w3.org/TR/sparql11-service-description/>

³<http://www.w3.org/TR/sparql11-entailment/>

⁴<http://www.w3.org/TR/sparql11-http-rdf-update/>

⁵<http://www.w3.org/2009/sparql/docs/fed/service>

⁶<http://www.w3.org/2009/sparql/docs/query-1.1/rq25.xml#propertypaths>

⁷<http://www.w3.org/2009/sparql/docs/csv-tsv-results/results-csv-tsv.html>

⁸<http://www.w3.org/2009/sparql/docs/json-results/json-results.xml>

JSON¹.

2.2 Setting the context: The Semantic Web and the Web of Linked Data

The current WWW is a web of documents and multimedia resources connected by hyperlinks. Following the same idea of interconnecting documents, the Web of Data consists of data resources of any kind connected by properties (links) using RDF. Tim Berners Lee suggests the following principles for the Web of Data [BL09]:

1. URIs should be used to identify things.
2. More specifically, HTTP URIs should be used so that people can look up things.
3. If someone looks up a URI, useful information should be provided using standards (RDF, SPARQL).
4. Further semantic links should point to other URIs so that people can discover more things.

The first rule indicates that things, resources, should be identified by URIs. Using the Universal Resource Identifier it is possible to provide a unique identifier for that object being then referenceable by other resources. The second rule is in line with the first rule, proposing the use of HTTP URIs for identifying data resources, since we are in a WWW context. Standards are the key element in the WWW, and also in the Web of Data. It is important that when we try to access data these data are understandable by everyone, and one way for achieving that is by using standards, such as the languages described in Section 2.1. Finally, the fourth rule promotes the discovery of new data by adding links to other data from the data resources, so that everyone can continue discovering new data.

Figure 2.4 represents the latest state of the LOD cloud (as of September 2011)². This figure depicts all the known RDF data sets that are exposed via SPARQL endpoints in the WWW. The topology of the LOD cloud is divided into six sets of data depending on the type of data: government data, life sciences data, media data, geographic data, user-generated content and cross domain data. All these data are exposed by means of SPARQL endpoints or RDF files and follow the principles for the Web of Data suggested by Tim Berners-Lee. Thus, it is possible to discover new data by following the links between the data sets. In the LOD cloud, one of the most referenced nodes is the DBPedia³ data set, which

¹<http://www.json.org/>

²<http://www4.wiwiw.fu-berlin.de/lodcloud/>

³<http://dbpedia.org/>

2. STATE OF THE ART

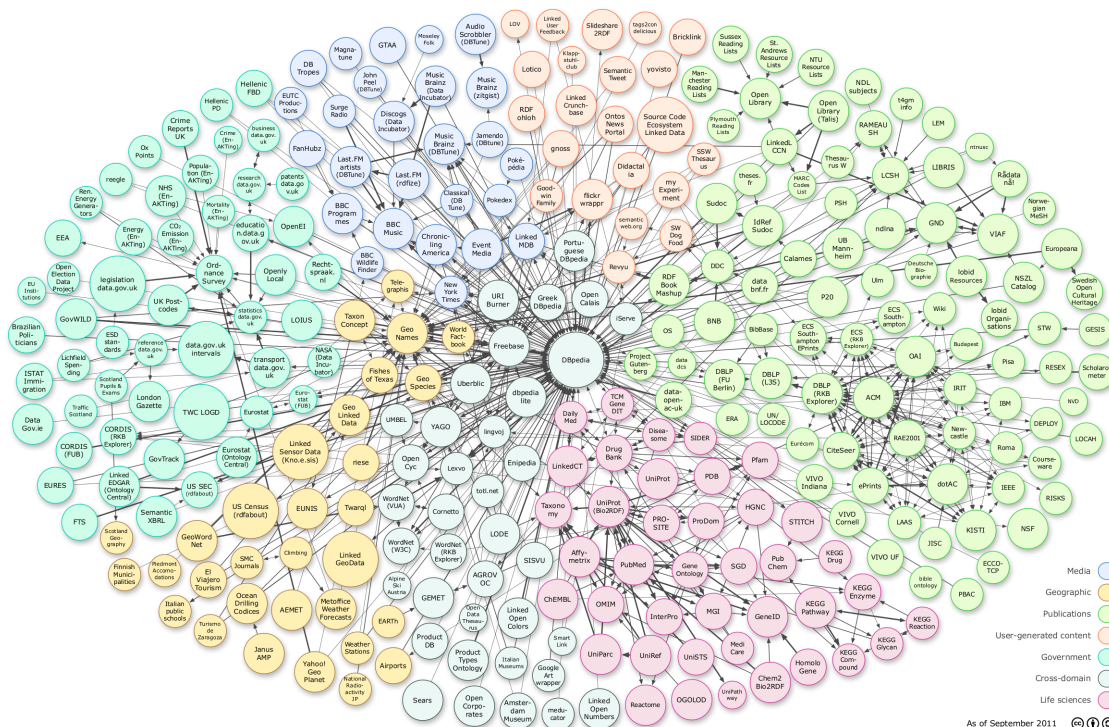


Figure 2.4: Linked Open Data cloud (as of September 2011)

appears at the centre of the cloud. DBpedia is the RDF representation of Wikipedia¹. Many other data sets contain links to it. For example, the `geo.linkeddata.es` data set, which contains Spanish geographic data, contains the resource `http://geo.linkeddata.es/resource/Provincia/Madrid`, which contains a link to the DBpedia resource `http://dbpedia.org/resource/Community_of_Madrid` which also contains links to other data sets in the cloud. A simple query using the SPARQL 1.1 Federation extension, which is described in Chapter 4 could query both data sets:

```
SELECT * WHERE {
  SERVICE<http://dbpedia.org/sparql> {
    ?resource rdfs:label "Community of Madrid" .
  }
  SERVICE<http://geo.linkeddata.es/sparql> {
    <http://geo.linkeddata.es/resource/Provincia/Madrid> owl:sameAs ?resource
  }
}
```

The previous query asks for the URI of a resource which is in both DBpedia and in `geo.linkeddata.es`. The following query asks in DBpedia for those countries that have an area greater than 20000 km and in El Viajero endpoint²:

¹<http://www.wikipedia.org/>

²<http://webenemasuno.linkeddata.es/sparql>

```

SELECT ?Book ?Country ?Area
WHERE {
  SERVICE <http://dbpedia.org/sparql> {
    ?Country rdf:type <http://dbpedia.org/ontology/Country> .
    ?Country <http://dbpedia.org/property/areaKm> ?Area
  } OPTIONAL {
    SERVICE <http://webenemasuno.linkeddata.es/sparql> {
      ?Book <http://webenemasuno.linkeddata.es/ontology/OPMO/refersTo> ?Country .
    }
  }
  FILTER(?Area < "20000"^^xsd:integer)
}

```

2.3 Data Integration and Query Federation

Data integration is defined as the process that involves combining data residing in different sources and providing users with a unified view of these data [Len02]. That is, data integration requires accessing distributed data across different data sources. Data integration can be accomplished via materialisation, federation and/or indexing. [Hul97] provides a classification of different architectures for integrating data:

- **Integrated read-only views: Mediation.** Mediation provides an integrated, read-only, view of data that resides in multiple databases. In addition to the source databases, a component called "mediator" is implemented, typically on a separate computer: that is, a mediator is defined as a system that supports an integrated view over multiple information sources [Wie92a]. A schema for the integrated view is available from the mediator, and queries can be made against that schema. The component may use different techniques to support the schema, including virtual and materialised views, and hybrids between them. The view supported by one mediator may involve data not also from a variety of information sources, but also from other mediators. The schema can be generated in two different ways, using a Global as View (GAV)[Lev01] or a Local as View (LAV)[Ull97] approach. In the GAV approach the global schema is expressed in terms of the data sources using views over the data sources. The views are created from each data source schema and then mapped against the global schema. When querying the global schema, this query has to be translated into the local schemas using the mappings created. In the Local As View approach [Ull97] the goal is the same as in GAV but the global schema is specified independently from the data sources. The relationships between the global schema and the sources are established by defining every source as a view over the global schema.
- **Multiple databases sharing information: Federation.** In contrast with data integration, federated architectures provide a framework whereby several databases can join in a federation. As mem-

2. STATE OF THE ART

bers of the federation, each database extends its schema to incorporate subsets of the data held in the other member databases. In most cases, a virtualised approach is supported for this approach [SL90].

- Integrated read-write views: Mediation with update. This architecture is an extension of the mediator architecture with update capabilities. The update functionality adds new challenges to the system like consistency and concurrency, which have to be managed by the architecture specification.
- Coordinated multiple databases: Workflow. Large organisations use large amounts of data repositories with hundreds of separate data repositories. These data repositories form an integral part of the enterprise, to help control in a variety of tasks. Workflows are also of key importance in mathematical models for life sciences or any other data intensive application. From the semantic perspective, the interaction between the databases can be characterised using the workflow paradigm.

2.3.1 Classification of Data Integration Approaches

There are several possible approaches to classify data integration. We have selected two approaches that are to be described in the following section: one that distinguishes between virtualised and materialised data integration, and other that distinguishes between procedural and declarative data integration.

2.3.1.1 Materialised and Virtualised data integration

As aforementioned, data integration can either be supported through virtual or materialised views. [CLNR98] provides a description of both data integration approaches. In the first case, which is typical of multi-databases, distributed databases, and more generally open systems, the integration system acts as an interface between the user and the source. In virtual integration query answering is generally costly, because it requires the application of query rewriting techniques and accessing the sources at query evaluation time. In the second case, the system maintains a replicated view of the source data. This is typical, for example, for both information systems re engineering and data warehousing. In materialised information integration, query answering is generally more efficient, because it does not require the accessing of the sources directly. However maintaining the materialised views is costly, especially when the views must be up-to-date with respect to any updates at the sources. There are many techniques for materialisation:

- Extract/Transform/Load (ETL) jobs extract data from one or more data sources, transform them as indicated in the job script, and then store the result in another data source.
- Replication makes and maintains a copy of data, often differentially by reading database log files.
- Caching captures query results for future reuse.
- Federation creates a virtual representation of the integrated set, only materialising selected portions as needed. It can be thought of as lazy integration.
- Search takes a different approach, creating a single index over the data being integrated. This is commonly used for unstructured data, and represents a partial materialisation, since typically the index identifies relevant documents, which will be fetched dynamically at the user's request.

2.3.1.2 Procedural and Declarative Data Integration

[CLNR98] also proposes a classification for data integration systems that considers procedural and declarative data integration. In the procedural approach, data are integrated in an ad-hoc manner with respect to a set of predefined information needs. In this case, the basic challenge is to design suitable software modules that access the sources in order to fulfil the predefined information requirements. They do not require an explicit notion of an integrated data schema, and rely on two kinds of software components: wrappers that encapsulate sources, converting the underlying data objects to a common data model, and mediators that obtain information from one or more wrappers or other mediators, re-define this information by integrating and resolving conflicts amongst the pieces of information from the different sources, and provide the resulting information either to the user or to other mediators. The basic idea is to have one mediator for every query pattern required by the user, and generally there is no constraint on the consistency of the results of different mediators.

In the declarative approach, the goal is to model the data at the sources by means of a suitable language, so as to construct a unified representation that can be used to query the global information system, and to derive the query answers by means of suitable mechanisms accessing the sources and/or the materialised views. This method allows the maintenance of a consistent global view of the information sources, which represents a reusable component of the data integration system.

2.3.2 Query Federation Systems

According to the classification in [Hu197], in our work we focus on query federation systems that follow a virtualised and declarative approach for data integration. These are a special case of federated database

2. STATE OF THE ART

management systems (FDBMS), which are defined as a collection of cooperating database systems that are autonomous and possibly heterogeneous, consisting of component DBSs that are autonomous yet participate in a federation to allow partial and controlled sharing of their data [SL90]. That is, there is no centralised control because the component DBSs (and their database administrators) control access to their data.

Some of the most relevant characteristics of FDBMS are identified in [SL90]:

- **Distribution:** Data may be distributed among multiple databases. These databases may be stored on a single computer system or on multiple computer systems, co-located or geographically distributed but interconnected by a communication system.
- **Autonomy:** DBSs are often under separate and independent control. Those who control a database are often willing to let others share the data only if they retain control. The authors identify the following autonomy types:
 - **Design autonomy:** this means the ability of a component DBS to choose its own design with respect to any matter. Heterogeneity in a FDBMS is primarily caused by design autonomy among component DBSs.
 - **Communication autonomy:** the ability of a component DBMS to decide whether to communicate with other component DBMSs.
 - **Execution autonomy:** the ability of a component DBMS to execute local operations (commands or transactions submitted directly by a local user of the component DBMS) without interference from external operations (operations submitted by other component DBMSs or FDBMSs) and to decide the order in which to execute external operations.
 - **Association autonomy:** a component DBS has the ability to decide whether and how much to share its functionality and resources with others

According to [SL90], FDBMS architectures can be loosely or tightly coupled architectures:

- In a **loosely coupled** FDBS each federation user is the administrator of his or her own federated schema. First, a federation user looks at the available set of export schemas to determine which ones describe the data he or she would like to access. Next, the federation user defines a federated schema by importing the export schema objects by using a user interface or an application program or by defining a multidatabase language query that references export schema objects. The user is responsible for understanding the semantics of the objects in the export schemas and

resolving the DBMS and semantic heterogeneity. In some cases, component DBMS dictionaries and/or the federated DBMS dictionary may be consulted for additional information. Finally, the federated schema is named and stored under the account of the federation user who is its owner. It can be referenced or deleted at any time by that federation user.

- In a **tightly coupled FDBS**¹ export schemas are created by negotiation between a component DBA and the federation DBA where the component DBA has authority or control over what is included in the export schemas. The federation DBA is usually allowed to read the component schemas to help determine what data are available and where they are located, and then negotiate for their access. The federation DBA creates and controls the federated schemas. External schemas are created by negotiation between a federation user (or a class of federation users) and the federation DBA who has the authority over what is included in each external schema. It may be possible to institute detailed and well-defined negotiation protocols as well as business rules (or some types of constraints) for creating, modifying, and maintaining the federated schemas.

In the context of query federation over the Web of Data, we can think of it as a loosely coupled FDBMS.

The most relevant components of a FDBMS architecture are:

- a) schemas which contain the descriptions of the data stored locally;
- b) mappings between the different database schemas, which can be defined as functions that correlate the schema objects in one schema to the schema objects in another schema; and
- c) processors, which are software modules that manipulate, transform, filter, access and construct data. These components can be combined in different ways to produce different data management architectures.

If we focus on schemas, [SL90] identifies five levels for managing the integration of the data in the system:

- The local schema represents the conceptual schema of a component DBS
- Component schema: A component schema is derived by translating local schemas into a data model called the canonical or common data model (CDM) of the FDBS.

¹For simplicity, we assume single (logical) federation DBA for the entire tightly coupled FDBS

2. STATE OF THE ART

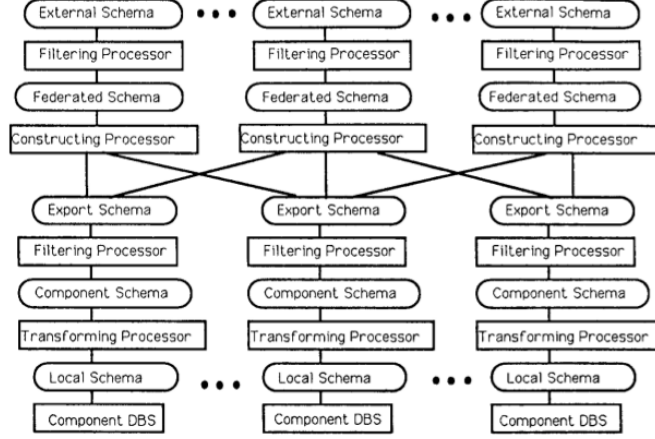


Figure 2.5: Reference architecture for a federated Database Management System [SL90]

- **Export schema:** Not all data of a component DBS may be available to the federation and its users. An export schema represents a subset of a component schema that is available to the FDBS. The purpose of defining export schemas is to facilitate control and management of association autonomy.
- **Federated schema:** A federated schema is an integration of multiple export schemas. A federated schema also includes the information on the data distribution that is generated when integrating export schemas.
- **External Schema:** An external schema defines a schema for a user and/or application - or a class of users/applications.

In Figure 2.5 the proposed architecture is represented.

2.3.2.1 Query Processing in Query Federation Systems

In this section we introduce the reader to the query evaluation and query processing systems firstly introducing a formal definition of those concepts. We base this formalisation in [P11].

A database schema S is a finite set R_1, \dots, R_k of relation symbols, with each R_i having a fix arity n_i . Let D be a countably infinite domain. An instance I of S assigns to each n-ary relation symbol R of S a finite n-ary relation $R^I \subseteq D^n$. The domain $dom(I)$ of instance I is the set of all elements that occur in any of the relations R^I . If a tuple a belongs to R^I we say that $R(a)$ is a fact in I . The set $Inst(S)$ is defined as the set of all instances of S . Given instances $I, J \in Inst(S)$, we write $I \subseteq J$ to denote that, for every relation symbol R of S , it holds that $R^I \subseteq R^J$.

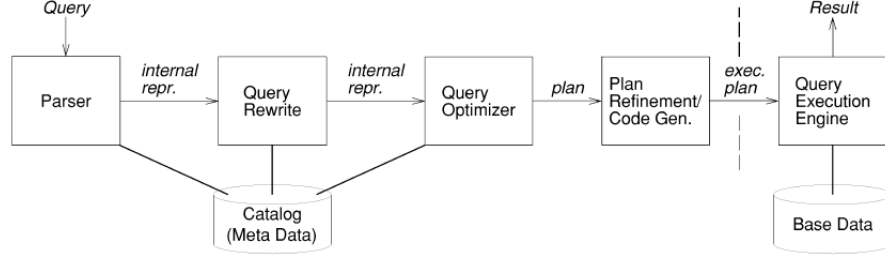


Figure 2.6: Most common phases of query processing

We define a k -ary query Q over a schema S , with $k \geq 0$, is a function that maps every instance $IInst(S)$ into a k -relation $Q(I) \subseteq dom(I)^k$. Notice that if $k = 0$ (Q is a Boolean query), then the answer to Q is either the set with one 0-ary tuple (denoted by true), or the empty set (false). Thus, if Q is a Boolean query, then $Q(I)$ is either true or false.

From a query perspective, [Kos00] presents a general architecture¹ (shown in Figure 2.6) with the following components: query parser, query rewriter, query optimiser, plan refinement component and query execution engine. The parser reads the query and transforms it into the systems internal representation. Next, the query rewriter creates a Logical Query Plan from this internal representation, which is the logical representation of a query. The query optimiser is in charge of applying different optimisations depending on the type of system, the physical state, the indices to use, the nodes to send the query to, etc. As a result, the query optimiser generates an optimised query plan that specifies how the query is going to be executed. This plan is refined and transformed into an executable plan by the plan refinement component. This plan will be executed by the query execution engine in each local node. The query execution engine provides generic implementations for every operator in the query plan. Finally, the catalogue (or metadata) component stores information about the databases (schema, tables, views or physical information about it), which can be used during parsing, query rewriting and query optimisation.

This architecture, can be adapted depending on the types of data sources that are handled, on the type of metadata available, etc. One of the most important elements in this architecture, since it heavily influences the performance of the query evaluation system, especially in query federation systems, is the query optimiser. In the next section we describe some of the most common optimisations that can be applied in such distributed settings. We start describing optimisations for query plans, then we move into query execution optimisations and finally we move into optimisation of specific client-server architectures.

¹In fact this architecture is generic to all kind of query processing systems, not only distributed query processors

2. STATE OF THE ART

2.3.2.2 Query Plan Optimisation in Distributed Query Processing

In this section we review some of the existing techniques for generating optimised query plans for Distributed Query Processing like deterministic algorithms (including dynamic programming algorithms and heuristics based algorithms) and randomised algorithms. We firstly introduce the dynamic programming [CLRS09] optimisation algorithm. The dynamic programming technique is an algorithm which is based on dividing a problem into simpler subproblems in a recursive manner. This algorithm is applicable when the subproblems are not independent, that is, when subproblems share subproblems. Next, these subproblems are solved just once, saving each solution in a table, and combining them to reach to the overall solution. Dynamic programming is based on the Principle of Optimality [Bel56] which states that a problem is said to satisfy it if the subsolutions of an optimal solution of a problem are themselves optimal solutions for their subproblems. For instance, the shortest path problem satisfies the principle of Optimality because if we have a graph with nodes a_1, \dots, a_n , and the shortest path from a_i to a_j is $a_i, x_1, x_2, \dots, x_m, a_j$ is a shortest path from node a_i to node a_j in a graph, then the portion of x_k to x_q on that path is a shortest path from x_k to x_q . In [CLRS09] authors define a dynamic programming algorithm as a four-step solution:

- Characterise the structure of an optimal solution.
- Recursively define the value of an optimal solution.
- Compute the value of an optimal solution in a bottom-up fashion.
- Construct an optimal solution from computed information.

In deterministic algorithms every algorithm builds a solution step by step in a deterministic manner. These algorithms apply either a heuristic or by exhaustive search. We present here a dynamic programming algorithm for distributed query processing and we also describe the use of heuristics for selecting the best possible query plan.

In a dynamic programming algorithm (like the one described in [Kos00] and depicted below) a plan can be discarded if an alternative plan exists that does the same or more work at a lower cost (Lines 3 and 10). The advantage of dynamic programming is that it produces the best possible plans depending on the costs of each plan: if these costs are sufficiently accurate, the algorithm will find the best possible query plan. The disadvantage is that it has exponential time and space complexity, thus, for complex queries, specially in a distributed environment, the complexity of dynamic programming may be prohibitive.

```

Input: SPJ query q on relations R1 , . . . , Rn
Output: A query plan for q
1: for i = 1 to n do {
2:   optPlan({Ri}) = accessPlans(Ri)
3:   prunePlans(optPlan({Ri}))
4: }
5: for i = 2 to n do {
6:   for all S in {R1 , . . . , Rn} such that |S| = i do {
7:     optPlan(S) = EmptySet
8:     for all O in S do {
9:       optPlan(S) = optPlan(S) UNION joinPlans(optPlan(O), optPlan(S - O))
10:      prunePlans(optPlan(S))
11:    }
12:  }
13: }
14: return optPlan({R1 , . . . , Rn })

```

Algorithms for selecting the best query plan using heuristics can also be used in distributed scenarios. In the generation of heuristics, the operator selectivity information can be used, although most probably, the use of heuristics will produce sub optimal query plans, since the best possible plans are produced by dynamic programming. Also, in RDF-based languages heuristics may be related to relationships among subjects, properties and objects, and other RDF characteristics, not only operator selectivity.

In a distributed scenario it is also necessary to take into account the costs of transmitting data between nodes. If a scan is performed in a node A, the data obtained from that scan will be transferred to another node in which an operation will be executed (i.e. a join).

In the previous algorithms, the following estimation models can normally be applied:

Cost Estimation for Plans The typical way to estimate the cost of a query plan is to estimate the cost of every operator in that plan to later on sum up all the individual costs [ML86]. The cost of each operation is defined by using cost metrics for each of them. Cost metrics can be extracted from the number of RDF triples that are read during query evaluation or the resource consumption. Resource consumption can be measured by their CPU consumption or the transmission costs. The costs can be weighted in order to model the impact of slow or fast machines and communication links. Transmission costs are calculated using fixed cost per transmission unit, per byte, etc.

Response Time Models This cost model considers the costs of each operator that is to be executed and also the costs of sets of operators executed in parallel. In parallel systems like [Gra90] queries are divided in sub queries that are processed in parallel whenever possible. [GHK92] proposes a cost model that measures the total resource consumption for each operator and the total usage of every shared resource used by a group of operators that run in parallel. The response time of an

2. STATE OF THE ART

entire group of operators that run in parallel is the maximum of the total resource consumption of the individual operators and of the total usage of all the shared resources.

Randomised algorithms could also be used for finding the most optimal query plan [SMK97]. In this type of algorithms a set of moves is defined. These moves constitute edges between the different solutions of the solution space; two solutions are connected by an edge if and only if they can be transformed into one another by exactly one move. Each of the algorithms performs a random walk along the edges according to certain rules, terminating as soon as no more applicable moves exist or a time limit is exceeded. The best solution encountered so far is the result.

2.3.2.3 Query Execution Optimisation in Distributed Query Processing

Independently of the cost estimation models previously described, there are distributed query execution techniques that can be applied in a distributed setting, specially focused on how to join data between distributed database tables and how the result data of these joins and other operators can be shipped from one node to another. The following techniques can be used for this purpose:

Row blocking Communication between distributed DBMS can generate a high load at the network.

Data is shipped between nodes using communication protocols that generate a high number of messages between nodes. The row blocking technique can be used to reduce this network overhead. This technique groups tuples that are shipped in blocks instead of individually, producing less messages and thus reducing the network overhead.

Optimisation of communication costs In a data federation scenario communication costs are important, thus, the network configuration is of key importance. Communication costs may vary significantly depending on the network configuration, depending on how far or how many nodes, the data has to go through. The optimiser has to select the cheapest way to send the data across the network nodes.

Multi-threaded Query Execution The use of several execution threads can be used to take advantage of query parallelisation. The optimiser can divide queries in multiple sub queries and assign threads to these sub queries to speed up query execution.[Gra90].

Joins with Horizontally Partitioned Data The logical properties of the join and union operators make it possible to process joins in a number of different ways if the tables are horizontally partitioned. If, for example, Table A is horizontally partitioned in such a way that $A = A1 \text{ UNION } A2$, then A and B can be computed in the following two ways [ESW78]:

- $(A1 \text{ UNION } A2).B$
- $(A1.B) \text{ UNION } (A2.B)$

The optimiser has to take into account all these considerations, although in an uncontrolled environment scenario such as that for the Web of Data this is not always applicable.

Semijoins The Semi join technique is a technique based on the idea of sending only those columns that are needed for doing the join operator over the network. Later on, the node will send only the rest of the tuples needed after the join has been performed.

Double-Pipelined Hash Joins The idea of this technique is that when a join between A and B is executed, two empty in-memory hash tables are created for A and B . First, tuples from A are processed by probing if its tuples (from A) are in hash table B . If a tuple from A matches the hash table then that tuple is immediately output and the inserted into hash table A for matching tuples from B that have not yet been processed. B tuples are processed analogously. The algorithm terminates when all A and B tuples have been processed and is guaranteed to find all the results of the join. However hash tables may grow up too much and overload main memory. To avoid that situation from arising, algorithms should perform some kind of memory backup.

Use of Bindings The use of bindings consists in using the results from a subquery directed to a specific database in the federation for restricting the next query to another federation. Imagine a query with an unconstrained remote query execution. This query execution may return more results than necessary. It may also happen that the database will not return all of them. Thus, an implementation of a query planner for federated queries may decide to decompose the query into two queries instead, where first the solutions from one query are obtained, and next dispatch to the remote database a constrained query with the solutions obtained from the query before. This optimisation may cause a high network overhead saturating the remote server if too many query results from one query are sent over the network.

Top and bottom queries Top and bottom queries are those queries that ask for the "five more expensive products in a store" or the "six least cited papers in a conference". Those queries generate extra data movement when the user only needs a small number of values. To avoid the unnecessary data transfer between nodes *stop* operators can be implemented. The federator specifies the remote database to compute the X top results, and then ship at most these X results.

2. STATE OF THE ART

Streaming results Data is streamed through between different nodes in the federation. One node starts producing data and sends a stream to another node that will start consuming it. Different nodes work on different parts of the data stream at the same time. This leads to more efficient execution times and reduced memory overheads.

In the following lines we focus on client-server architectures. We describe main characteristics of these types of architectures as well as providing a classification for them.

2.3.2.4 Query Federation Architectures

In this section we focus on the architecture of federated systems. We describe how resources are used within the federation and we also describe specific query optimisation techniques in this type of architectures. The following architectures for database processing can be used:

Peer-to-peer Each site acts as a server in the federation and stores parts of the database. Sites can also act as clients and send queries to the federation. In this architecture policies can be defined so that not all sites can communicate with others.

Client-server In a strict client-server architecture, every site in the federation has a fixed role of always acting as a client (query source) or as a server (data source). In the federation, not all clients may be able to communicate with other clients. Servers may not communicate with other servers either.

Middleware, multitier A hierarchy of sites is created where each site can act as server or client, depending on its place in the federation and the site that is communicating with it. Sites act as servers for the lower level nodes and as clients for the upper level nodes. A site can only communicate with the nodes in above level playing the client role and with the nodes in the lower level playing the server role. A site cannot communicate with sites at the same or any other level.

In a client-server architecture databases are stored by server machines with more computing resources than clients. This places important questions related to communication costs between servers and clients: where will it be best to execute a query? Is it useful to transfer the data from the servers to the client machines or will be better to move the query to the data assuming all associated communication costs? In this context several alternatives appear:

Query shipping The idea is to send queries from clients to servers, executing them at the lowest level possible in a hierarchy of sites. A client sends the query to the server which will remotely evaluate

it, finally shipping back the results to the client. In a multiple server configuration, query shipping will only work if there is a middle layer (with server or gateways) that transfer results between servers and clients.

Data shipping The idea is to execute queries at client sites, shipping data from servers to the clients. Data is cached at client machines in main memory or on disk.

Hybrid shipping Hybrid shipping is a mix of both data and query shipping. Query operators are executed on client or server machines, depending on the optimisation techniques, allowing caching data by clients.

Client-server systems can be optimised using the specific characteristics of their architectures. We describe in here the query optimisation techniques for the systems implementing these architectures.

Site selection The site selection technique defines at which site a specific operator from the query plan is going to be executed. This selection of the execution node will define the data transfer pattern: data shipping, query shipping, and hybrid shipping can be modelled by the options they allow for site selection. Each operator in the logical query plan has a (logical) site annotation, indicating where the operator is going to be executed. The site selection will depend on the server's characteristics, network latency, amount of data to be transferred, etc. This selection can be done using heuristics or cost based algorithms.

Optimisation place The optimisation can take place at different nodes. The node selection for creating the optimised query plan is important due to the knowledge of the different nodes in the federation that one node can have. One option for selecting the site where the query will be optimised is to do the query parsing and query rewrite out at the client whereas query optimisation and plan refinement could be carried out at the server [HF86]. This approach provides more knowledge about the current state of the system (i.e., the load on the server) and should, therefore, be carried out by the server. In a multi server configuration there are no servers with complete knowledge of the whole system. In this configuration a server should have enough knowledge of the federation (other server states and loading, network latency, etc.) for generating the best possible query plan. Asking other servers about their state is also a possibility for gaining knowledge about the current federation status but it also involves at least two extra messages for every server that is potentially involved in a query.

2. STATE OF THE ART

When to optimise a query There are several possibilities at which a query can be optimised. One possibility is to optimise a query at system's compile time, depending on the information about the data nodes available. The downside of this approach is that when something happens (i.e. a server is down), the query plan will fail entirely. A solution to this problem is to optimise queries on the fly. The execution of the plan is started with a compiled or dynamically chosen plan. Next, the execution of this query plan is observed and if something is not working correctly or if the expectations are not met, the query plan is altered. Intermediate results are materialised, and the optimiser will try to find another query plan for those parts of the plan that failed. Another dynamic query optimisation technique divides the execution and the optimisation in two different phases. In the first phase the query is decomposed into a set of subqueries that can be executed by a single server. This allows to parallelise these single queries, composing the final query result by joining the results of the subqueries. The speed of the servers producing the results for each query is based on the selectivity and the cost of the joins which need to be carried out to combine the subquery results. The goal is to parallelise work at the client with work at slow servers as much as possible and to avoid the execution of very expensive joins that may result from poor join ordering.

Two step optimisation The two step optimisation consists of first generating a plan that specifies the join order, join methods and access paths at compile time. Next every time before the query is executed, the plan is transformed to determine where every operator is to be executed. Both steps can be carried out by dynamic programming or any other enumeration algorithm.

2.3.3 Challenges and Limitations of Traditional Federated Query Processing Technology

The challenges for query optimisation, specially in the Web of Linked Data are the following:

Cardinality estimations Statistics on the Web may not be reliable due to the changing environment (and the lack of updated statistics or which may not exist at all). Since the cost estimation process depends on these cardinality estimations, query planners should adapt their optimisation strategies to these situations. Correlations between predicates can cause intermediate result cardinality estimates be very different from reality.

Data and environment dynamics The query environment characteristics may be changing dynamically. On the Web data may not change very rapidly but in other environments like data streams [CCG10], queries might need long execution times, and the data characteristics might change

during the query execution along with run time costs. Server workload may also vary due to the number of queries submitted. These situation make it necessary to adapt queries.

Complex queries involving many SPARQL endpoints Query optimisers typically switch to a heuristic approach when queries become too complex to be optimised using the dynamic programming approach. Such queries are naturally more prone to estimation errors [IC91], and the use of heuristics exacerbates the problem.

Interactive querying When the querying environment is very interactive (such queries on the Web, in which users may query for different data) the optimise-then-execute model does not fit well. The main problem in this situation is that cardinality estimates change too quickly for the optimisation. Also, pipelined execution and early-result scheduling, even in the presence of slow data sources, becomes very complex.

These challenges generated research in Adaptive Query Processing [DIR07] (AQP), creating new operators and techniques for adapting query plans to the evolving situation. In the following lines we present the basics of Adaptive Query Processing, describing basic operators, environments and behaviour of these operators.

2.3.4 Adaptive Query Operators

Traditional query operators do not adapt themselves to specific conditions of the query execution. In the next few lines we present the basic adaptive query operators that allow the query executor to adapt to the specific conditions of the query execution:

Symmetric Hash Join Operators The traditional hash join operator waits for the data to arrive before it starts processing and producing results. In a stream-based architecture this is not the best configuration and the operator has to be reconsidered. Besides, this way of employing the join operator makes difficult the query adaptation in case that input data becomes unavailable due to network traffic or lost of communications. Symmetric Hash Join stores tuples when they are read in a hash table and these are probed against the opposite table. The operator processes data from either input, depending on availability. This operator also considers frequent moments of symmetry [AH00] - points at which the join order can be changed without compromising correctness or without losing work. The main disadvantage of this join operator is its high memory usage since a hash table must be built on the larger input relation as well.

2. STATE OF THE ART

Eddy [AH00] The basic idea is to treat query execution as a process of routing tuples through operators, and to adapt by changing the order in which tuples are routed through the operators. The operator, which is used as the tuple router, monitors the execution, and makes the routing decisions for the tuples.

n-ary Symmetric Hash Joins/MJoin This operator is a generalisation of the Symmetric Hash Join but generalising it to multiple inputs [RRD⁺03]. The operator treats the input relations symmetrically and by allowing the tuples from the relations to arrive in an arbitrary interleaved fashion. The operator builds a hash index on every join attribute of every relation in the query and uses a lightweight tuple router to route the tuples from one hash table to another. When a new tuple from a relation arrives into the system, it is first built into the hash tables on that relation, and then the hash tables corresponding to the remaining relations are probed in some order to find the matches for the tuple. Memory consumption is an important problem of this operator, since a memory index has to be built for each input join.

Adaptive Query Processing addresses problems that are not present in traditional (and distributed) query processing systems which assume that statistics are present, servers are available, costs are not modified during execution time, etc. Adaptive query processing faces the problems generated from eventually not having some of the previous information: missing statistics, unexpected correlations, unpredictable costs, and dynamic data by using feedback to tune execution. AQP is also generalisable to many other contexts, particularly at the intersection of database query processing and the Web.

2.3.5 SPARQL optimisations

Most of the existing optimisations for SPARQL focus on isolated RDF databases since the federation of SPARQL queries is relatively new. New optimisations have been created, mainly for distributed SPARQL query processing. For instance, authors in [AVL⁺11] propose the use of adaptive query techniques in distributed SPARQL queries. They propose an adaptive version of a join operator extending the Xjoin [UF00] and gjoin [VRL⁺10] operators. This new operator called agjoin is a non-blocking operator that extends a symmetric hash join operator by allowing parts of the hash tables to be moved to secondary storage. They also implement the delay-hiding feature Query Scrambling [UFA98]. This adaptive technique modifies query execution plans on-the-fly when delays are encountered during run time.

We consider specific SPARQL optimisations those that are based on using characteristics and the properties that are provided by the SPARQL query language. We identify mainly SPARQL pattern

reordering and SPARQL pattern grouping for federating queries. Even though it is not itself an optimisation but a characteristics of some SPARQL federated query processors, we consider as an optimisation to provide a list of SPARQL endpoints for submitting a query.

Semantic SPARQL Query Optimisation [MSL10] define semantic SPARQL query optimisation (SQO) as constraint-based optimisation. The idea of SQO is to exploit integrity constraints over the input database. These constraints restrict the state space of the database and often can be used to rewrite queries into equivalent, but more efficient, ones. Constraints could be user-specified, automatically extracted from the underlying database, or may be implicitly given by the semantics of the RDFS vocabulary.

SPARQL Query Pattern Reordering We define pattern reordering as a reorder of SPARQL patterns (OPTIONAL, JOIN, FILTER, etc.) without the need of statistics to obtain a gain in the performance of the query.

SPARQL Pattern Grouping Pattern grouping consists in the identification of subsets of SPARQL triple patterns in a query that can be directed to the same SPARQL endpoint. The identification of these common triple patterns is based on their predicates and a list of endpoints to which to direct these patterns.

Source Ranking [LT10] proposes a classification of sources in a ranking. A source is relevant if it contains data that can contribute to the final answers. The standard optimisation goal is to (1) obtain all results as fast as possible. However, given the volume and dynamic of the Linked Data collection, it is often unfeasible to retrieve and process all sources. It is important to rank sources by their relevancy to the query and more fine-grained optimisation goals. In particular, it might be desirable to (2) report results as early as possible, (3) to optimise the time for obtaining the first k results, or (4) to maximise the number of total results, given a fixed amount of time.

In this section we do not consider other SPARQL query optimisation techniques since they are more focused on RDF triple stores. Some of these techniques are the use of a main memory index proposed by Hartig [HH11] (which is also similar to the adaptive Eddy approach [AH00]), partition vertically RDF data grouping RDF triples with the same property [AMMH09], the use of quad patterns [HD05], disk-based data structures for RDF data [WKB08, NW08], etc.

2.4 Ontology-Based Data Integration

In the previous section we saw different approaches for integrating data. In this section we focus on integrating RDF data through distributed query processing. We assume that the common data model is RDF, and thus no mapping is necessary to integrate all data sources. Querying will be performed by Query Federation Systems, accessing different RDF data sets.

2.4.1 Distributed SPARQL Query Processing Systems

Some of the existing engines supporting the official SPARQL 1.1 federation extension are ARQ¹, RDF-Query², Rasqal RDF query Library³ or ANAPSID[AVL⁺11]. There are also other systems that implement a distributed query processing system for SPARQL like DARQ [QL08], NetworkedGraphs [SS08], ADERIS [LKMT11], FedX [SHH⁺11] SPLENDID [GS], [LT11] or SemWIQ [Lan08] but they do not follow the SPARQL 1.1 Federation specification. Other system that supports distributed RDF querying is presented in [HSB04]. We do not consider this system here as it uses the query language SeRQL [JB03] instead of SPARQL.

ANAPSID implements two adaptive query operators (agjoin and adjoin operators). The agjoin operator uses a hash join along with storing join tuples for speeding up join operators. The adjoin operator hides delays coming from the data sources and perform dereferences for certain predicates.

ADERIS decomposes federated SPARQL queries into multiple source queries and integrates the results utilising two techniques: adaptive join reordering, for which a cost model is defined, and the optimisation of subsequent queries to data sources to retrieve further data. For optimising these queries ADERIS implements a greedy algorithm so it is possible to find the best possible query plan. Joins are index nested loop joins using an index on join attributes to look up matching tuples from the right input and output joined tuples to the next operator in the plan. They also use a technique for reordering pipelined index nested loop join-based query plans based in [LSM⁺07].

Ladwig et al.[LT11] implements a new Join operator, called Symmetric Index Hash Join (SIHJoin), which combines queries to remote SPARQL endpoints with queries to local RDF data stores. When this situation happens, data retrieved from the local RDF data set is stored in an index hash structure for faster access when performing a join with remote data. The authors also provide cost models for their previous work of using non-blocking operators for joining data [LT10].

SPLENDID is a query federation system which extends Sesame⁴ adding a statistics-based join re-

¹<http://jena.sourceforge.net/ARQ/>

²<http://search.cpan.org/dist/RDF-Query/>

³<http://librdf.org/rasqal>

⁴<http://www.openrdf.org/>

ordering system. SPLENDID bases its optimisations in join reordering rules based in a cost model described in [GS]. The statistics are collected from VoID descriptions and allow to perform join reordering in an efficient manner.

SemWIQ is a mediator-wrapper based system, where heterogeneous data sources (available as CSV files, RDF data sets or relational databases) are accessed by a mediator through wrappers. Queries are expressed in SPARQL and consider OWL as the vocabulary for the RDF data. SemWIQ uses the Jena's SPARQL processor ARQ to generate query plans and it applies its own optimisers. These optimisers mainly consist of rules to move down filters or unary operators in the query plan, together with join reordering based on statistics. The system has a registry catalogue that indicates where the sources to be queried are and the vocabulary to be used.

DARQ is a SPARQL query federation system, which also extends the Jena SPARQL processor ARQ. This extension requires attaching a configuration file to the SPARQL query, with information about the SPARQL endpoint, vocabulary and statistics. DARQ applies logical and physical optimisations, focused on using rules for rewriting the original query before query planning (so as to merge basic graph patterns as soon as possible) and moving value constraints into subqueries to reduce the size of intermediate results. Other important drawback of DARQ is that it can only execute queries with bound predicates. Unfortunately DARQ is no longer maintained.

Networked Graphs creates graphs for representing views, content or transformations from other RDF graphs, and allows the composition of sets of graphs to be queried in an integrated manner. The implementation considers optimisations such as the application of distributed semi-join optimisation algorithms.

FedX [SHH⁺11] also extends Sesame and bases its optimisations in grouping joins that are directed to the same SPARQL endpoints and rule-based join optimiser, which orders a list of join triple patterns (or groups of these triple patterns) according to a heuristics-based rule, preferring join groups. FedX also reduces the number of intermediate joins by grouping sets of mappings in a single subquery using SPARQL UNION constructs and sending this subquery to the relevant data sources.

Hartig and colleagues[HBFO9] propose model that tries to exploit the navigational structure of the Web of Data, by incrementally executing queries over it. They discover new URIs from the initial SPARQL query and populate a local RDF repository, which is queried again for new answers to the initial query. In a more recent job from Hartig [Har11] the author proposes a heuristic for identifying a suitable order for query execution as well as using main memory index described in [HH11].

2. STATE OF THE ART

2.4.1.1 Discussion on Query Federation System

The following tables summarise what characteristics the systems for federating SPARQL queries have. These characteristics are extracted from Section 2.3 which contains the main properties that distributed query processing systems have.

Table 2.1: Data Integration Level

	FedX	ANAPSID	SPLENDID	ARQ	RDF::Query	NetworkedGraphs	SemWIK	DARQ	ADERIS	SIHJoin
Mediation	No	No	No	No	No	No	Yes	No	No	No
Federation	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes
Mediation with update	No	No	No	No	No	No	No	No	No	No
Data Workflows	No	No	No	No	No	No	No	No	No	No

Table 2.1 summarises the type of architecture for information/data integration that are implemented by the distributed SPARQL query processors. Most of them implement a federated architecture, since almost all of them focus in just querying a set of remote RDF databases. The only one that implements a mediator architecture is SemWIK which integrates information from several sources.

Table 2.2: Type of client-server System's Architectures

	FedX	ANAPSID	SPLENDID	ARQ	RDF::Query	NetworkedGraphs	SemWIK	DARQ	ADERIS	SIHJoin
Client-server	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Middleware	No	No	No	No	No	No	No	No	No	No
P2P	No	No	No	No	No	No	No	No	No	No

Table 2.2 presents the type of client-server architecture is implemented by each of the systems. All implement the strict client-server architecture: every site in the federation has a fixed role of always acting either as a client. All systems act as clients and queries are sent to remote servers.

Table 2.3: Query Planning Optimisation Techniques

	FedX	ANAPSID	SPLENDID	ARQ	RDF::Query	NetworkedGraphs	SemWIK	DARQ	ADERIS	SIHJoin
Heuristics-based	Yes	Yes	No	No	Yes	No	No	No	Yes	Yes
Dynamic Programming	No	No	No	No	No	No	No	No	No	No
Cost estimation	No	No	Yes	No	No	No	Yes	Yes	Yes	No
Response-time models	No	No	No	No	No	No	No	No	No	No
Randomised Algorithms	No	No	No	No	No	No	No	No	No	No

Table 2.3 presents the techniques used by the systems for generating the most optimal query plan. SPLENDID, SemWIK and DARQ implement the use of cost-based algorithms for selecting the best

query plan. All of them gather statistics from each node and calculate the best possible execution of the query plan. FedX uses a rule-based join optimiser, which orders a list of join arguments according to a heuristics-based cost estimation. Also, other systems implement heuristics-based algorithms to choose the best query plan. None implement neither a dynamic programming algorithm nor a randomised algorithm.

Table 2.4: SPARQL Specific Features

	FedX	ANAPSID	SPLendid	ARQ	RDF::Query	NetworkedGraphs	SemWIK	DARQ	ADERIS	SIHJoin
SPARQL 1.1 Fed	No	Yes	No	Yes	Yes	No	No	No	No	No
Semantic Query Optimisation	No	No	No	No	No	No	No	No	No	No
Pattern reordering without statistics	No	No	No	No	Yes	No	Yes	No	No	No
Pattern Grouping	Yes	No	No	No	No	No	No	No	No	No

Table 2.4 shows what systems implement SPARQL specific optimisations or features. It shows that the systems are divided in two main groups: those systems that implement the SPARQL 1.1 Federation extension and those that do not implement it, but implement a source selection technique. Among the systems that do not implement the official specification all of them implement some kind of optimisation. The most common optimisation is the cost based optimisation. Systems implementing cost based optimisations are NetworkedGraphs, SemWIK, SPLendid, DARQ and the latest version of FedX. This optimisation requires statistics from the data sets for each predicate, which can then be easily applied to the SPARQL query. This optimisation could be not as easily applied in those systems that implement the SPARQL 1.1 since triple patterns are grouped and statistics should be over the whole query to the remote SPARQL endpoint, not for each predicate.

FedX provides a pattern grouping optimisation, grouping triple patterns that are directed to the same SPARQL endpoint. FedX contains a list of available endpoints which is initialised by issuing ASK queries to each endpoint. When the first query is submitted to FedX along with the list of SPARQL endpoints to query, an ASK query is submitted to all the SPARQL endpoints in that list for identifying where to send each triple pattern. This is the only system that applies such optimisation using ASK queries. This pattern grouping is based on each predicate, having a list of possible SPARQL endpoint that contains such predicate.

Pattern reordering is only implemented by RDF::Query and SemWIK. The former provides a way for inserting FILTERs in the SERVICE invocation when possible while the latter reorders FILTERs depending on the query plan generated, moving them to the lowest part of that query plan.

2. STATE OF THE ART

Table 2.5: Query Execution Optimisations

	FedX	ANAPSID	SPLendid	ARQ	RDF::Query	NetworkedGraphs	SemWiq	DARQ	ADERIS	SIHJoin
Row blocking	No	Yes	No	No	No	No	No	No	No	No
Multicast optimisations	No	No	No	Yes	Yes	No	No	No	No	No
Multi-threaded	Yes	No	No	No	No	No	No	No	No	No
Horizontally Partitioned Data	No	No	No	No	No	No	No	No	No	No
Semi-joins	No	No	No	No	No	No	No	No	No	No
Double-Pipelined Hash Joins	No	Yes	No	Yes	Yes	No	No	No	No	No
Use of BINDINGS	Yes	No	No	Yes	Yes	No	No	No	No	No
Top-Bottom Optimisations	No	No	No	No	No	No	No	No	No	No
Streaming	No	No	No	No	No	No	No	No	No	Yes

Table 2.5 shows the optimisations described in 2.3.2.3. These optimisations focus on the query execution phase. Only FedX offers a parallelisation system. For quickly execution FedX uses a multi threaded worker pool to execute the joins, and union operators in a highly parallelised fashion. They use a pipelining approach such that intermediate results can be processed in the next operator as soon as they are ready yielding higher throughput. The use of a BINDINGS query is only used by ARQ and FedX. The use of this operator consists in first issuing a query to a SPARQL endpoint, and then using the results of the query to restrict the results for the next queries to be sent to other SPARQL endpoints. FedX uses a SELECT UNION pattern, that is, using the results of one query, it generates a UNION query for each of the results that are to be sent to the remote endpoint. ARQ uses a similar approach, but without the UNION pattern. For queries that produce small amounts of results this approach works OK, but for large amounts of results, the overhead produced in the remote server may cause it to reject incoming connections. Hash Joins, as already summarised, are present in ADERIS, ANAPSID and SIHJoin, providing an join based on an existing index for speeding up the results generation. Streaming is provided by ADERIS and SIHJoin where source data is treated as finite streams that can arrive at any time and in any order.

The main difference with the systems described before is the use of the SPARQL 1.1 Federation extension and therefore the compatibility with its semantics. Results may be different between systems since they use different semantics, specially if there are systems that redirect one query to several endpoints. All these results have to be joined with the previous results and thus, different results will be generated.

Table 2.6 shows whether the systems choose to ship the query plan to be executed at the server side or whether the query is processed in the client's processor and what is shipped is the data from the

Table 2.6: Query or Data Transfer

	FedX	ANAPSID	SPLendid	ARQ	RDF::Query	NetworkedGraphs	SemWIK	DARQ	ADERIS	SIHJoin
Query shipping	No	No	No	No	No	No	No	No	No	No
Data shipping	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Hybrid-shipping	No	No	No	No	No	No	No	No	No	No

servers. All systems use the latter approach.

Table 2.7: Optimisation in client-server Architectures

	FedX	ANAPSID	SPLendid	ARQ	RDF::Query	NetworkedGraphs	SemWIK	DARQ	ADERIS	SIHJoin
Site selection	No	No	No	No	No	No	No	No	No	No
Optimisation place	Client	Client	Client	Client	Client	Client	Client	Client	Client	Client
Optimisation time	Compile	Compile	Compile	Compile	Compile	Compile	Compile	Compile	Compile	Compile
2-step optimisation	No	Yes	No	Yes	Yes	No	No	No	No	No
Non Blocking operators	No	No	No	No	No	No	No	No	No	No

Table 2.7 shows at what time systems perform the optimisation. All of them perform the optimisation at compile time and do not modify this query plan later on in processing time.

Table 2.8: Adaptive query optimisations

	FedX	ANAPSID	SPLendid	ARQ	RDF::Query	NetworkedGraphs	SemWIK	DARQ	ADERIS	SIHJoin
Symmetric hash joins	No	No	Yes	No	No	No	No	No	Yes	Yes
Eddi	No	No	No	No	No	No	No	No	No	No
N-ary hash joins	No	No	No	No	No	No	No	No	No	No

Finally, in Table 2.8 is represented what systems implement adaptive query processing techniques. They are only applied by ANAPSID, ADERIS and by [LT11]. ANAPSID implements a version of a hash join operator which are non blocking blocking operators. ADERIS implements a version of a hash-join operator, plus other cost-based operators. The SIHJoin proposed by [LT11] is also a type of adaptive operator since it also manages network problems by using non blocking techniques.

2.4.2 RDB2RDF systems

It may be the case that SPARQL endpoints virtualise other data sources like RDBMS, spreadsheets, etc. We will focus on RDB2RDF systems. RDB2RDF systems [SHH⁺09] map relational databases to RDF, providing semantic access to existing relational databases by mapping relational entities to RDF classes and properties. There are two features that characterise these systems:

2. STATE OF THE ART

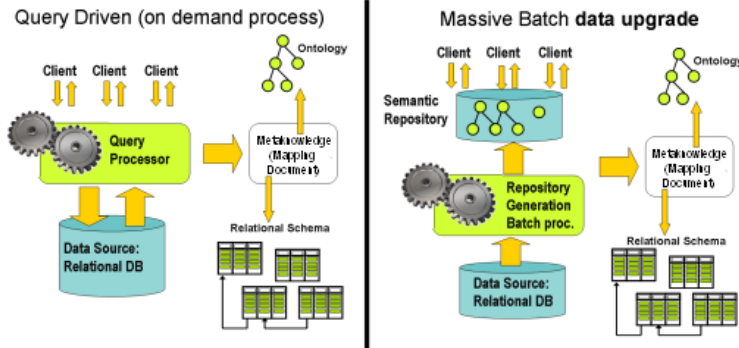


Figure 2.7: RDB2RDF systems working modes

- Mapping creation: mappings can be created automatically, normally resembling the initial database structure, or manually, normally driven by a set of domain ontologies.
- Materialisation vs. virtualisation of databases: these systems can normally operate in two different modes: materialising (dumping) the database into RDF, and vitalising the database, by using RDF query languages directly without creating any instance in the ontology.

Figure 2.7 depicts these two working modes. In both cases, the query processor interprets the mappings and accesses the databases according to them.

There are many systems that map relational databases to RDF (R2O[RP06], D2R[BC07], etc.). We will describe one of them, R2O, and its associated processor. A more exhaustive list can be found in [SHH⁺09].

2.5 Conclusions

In this chapter we surveyed the current state of the art in integrating data using distributed query processing systems. Such systems presented two main problems: heterogeneity of data sources and access to distributed data sources. The problem of the heterogeneity of data sources is partially solved by the usage of a common data model. The Resource Description Framework is the proposed model by the W3C for integrating heterogeneous data on the Web. Currently there is an important growth of available data sets that are exposed on the Web using the RDF data model. In Figure 2.4 different data sets from different domains are represented, from government data to biological data, all distributed and available via the HTTP protocol.

The distributed query processing. We showed systems that provide solutions for distributed query processing, (when talking about databases these systems are known as database management systems).

These systems allow querying a set of data sources as if they were a single data source. Queries are sent by clients and then optimised and distributed across the data sources. Many research problems arise here as well: query translation from the client's schema to each data source schema, local or remote query execution, data shipping, query optimisation taking into account the previous problems, etc.

In this thesis we focus on Distributed Query Processing, more specifically in optimisation of distributed query processing for RDF data. In this section we saw that there are not many systems that distribute and optimise queries to RDF data sources using SPARQL. The first system for accessing distributed RDF data sets using SPARQL is DARQ [QL08], which provided basic pattern reordering based on previously gathered statistics. The next systems also used statistics ([SS08, Lan08]) also provided pattern reordering using statistics and other optimisation techniques like pushing down filters as much as possible. They also have in common that they base their optimisations in the triple patterns predicates, for selecting the data sources to which redirect remote queries. Next, the W3C started developing the SPARQL 1.1 Federation Extension, which provided a common vocabulary for federating queries to remote SPARQL endpoints (currently in Last Call period), allowing users to direct queries to remote SPARQL endpoints. In despite of that, other systems continued basing their implementations in the use of triple pattern predicates for identifying data sources like [SHH⁺11, LKMT11, GS] while others implemented this specification like [AVL⁺11], all of them providing different optimisation techniques.

The optimisation techniques implemented for federated SPARQL query processing focused mainly in the use of statistics for reordering join patterns ([QL08, Lan08, GS, SS08]), adaptive query processing techniques ([AVL⁺11, LKMT11, LT11]) or grouping triples patterns for submitting them to the same SPARQL endpoints (besides of parallelising these queries and the use of BINDINGS) like [SHH⁺11]. However, none of these systems provided an analysis of the usage of the OPTIONAL pattern in the new distributed scenario, which is the most costly operator in SPARQL [MSL10]. Besides, a formal study of the new SPARQL 1.1 Federation Extension does not exist in the current state of the art. Apart from [AVL⁺11], none of the systems focused on the processing of large amounts of data, focusing only on querying SPARQL endpoints that limit the number of returned results.

In the next Chapter, we provide a more detailed description of these problems, and also provide the goals, hypothesis and research problems of this thesis.

2. STATE OF THE ART

3

Objectives and contributions

3.1 Objectives

In this chapter we present the goals of our work, together with the open research problems that we aim to solve. Besides, we detail the contributions to the current state of the art, the work hypotheses, the assumptions considered as a starting point for this work, the restrictions of the work and the results obtained are presented.

3.1.1 Goals and Research Problems

The goal of this thesis is to define access mechanisms to distributed RDF data sources. In the context of this research, we identify three main sub-goals. First, we **provide formal semantics for the SPARQL 1.1 Federation Extension**; second, we **propose optimisation techniques for the evaluation of SPARQL queries in distributed settings**. This second sub-goal can be divided further into two sub-goals: **to identify specific characteristics of the SPARQL language for querying the RDF data model that may allow optimising distributed queries** and **to identify and adapt existing optimisation techniques used for distributed query processing for the optimisation of SPARQL queries**. The third sub-goal of this thesis consists on **proposing a test suite for measuring the performance of Distributed SPARQL Query Processing systems**.

In order to achieve the first goal, the following research problems must be solved:

- The problem of defining the syntax and semantics for the SPARQL 1.1 Federation Extension is still open. This specification document by the W3C is in a Last Call Working Draft status, meaning that work still has to be done in defining specific operators semantics. The definition of its semantics is problematic, specially in the SERVICE VAR pattern evaluation, due to the

3. OBJECTIVES AND CONTRIBUTIONS

evaluation semantics of the main SPARQL 1.1 query document. In [PAG09] authors proposed a formalisation of the syntax and semantics for the SPARQL 1.0 query language, which is the basis for the SPARQL 1.1 query document. We use [PAG09] as the basis for our research and extend it by providing the formal syntax and semantics for the SPARQL 1.1 Federation Extension.

- The formalisation of the SPARQL 1.1 Federation Extension leads to new evaluation problems for a SPARQL query. These problems concentrate on the evaluation of the SERVICE VAR pattern, since an execution order for the SPARQL query must be proposed in order to guarantee its correct execution. We provide an analysis of these evaluation problems and a solution for them.

The second goal of our work is **to optimise the access to distributed RDF data sets**. This goal is also an open research problem far from being solved. Current systems, as we have seen in Chapter 2, implement few optimisation methods. Most of the research in this area started several months ago and results are still at an early stage. For optimising the access to distributed RDF data sets, we identify two sub goals that will help us make advances in this research problem.

The third goal (**proposing a test suite for measuring the performance of the different Distributed SPARQL Query Processing systems**) has the following associated research problems.

- The current SPARQL benchmarks (Berlin SPARQL Benchmark [BS09], SP²Bench [MSP10] and Fedbench [SGH⁺11]), are not appropriate for our purposes. The first two are not designed for a distributed environment, while the third is based on a federated scenario that is not as comprehensive as the Berlin SPARQL Benchmark and SP²Bench. Thus, an improvement of these benchmarks or a proposal for benchmarking distributed SPARQL query processing systems is needed.
- The proposal of a benchmarking system has a set of problems associated, such as a formal study of the characteristics of the language that will be used. This formal analysis has already been carried out in the previous research goals. Next, is to identify a broad set of queries that will allow us to measure how the systems that are to be benchmarked behave with each specific query.

The previous methodological goals have two associated technological goals. The first goal is **the implementation of a federated SPARQL query evaluation system (SPARQL-DQP)**. The implemented system extends OGSA-DAI [Ant07] and OGSA-DQP [ea09] with a new data resource (RDF) and a new query language (SPARQL). The next technological goal is **to optimise the access to these RDF data resources by implementing the optimisation techniques to access distributed RDF data proposed**

in the methodological objectives. In this goal we implement the optimisation techniques that were identified by the formal analysis of the SPARQ 1.1 Federation Extension specification document.

3.2 Contributions to the State of the Art

In this work, we aim to provide solutions to the previous open research problems. Chapter 4 describes the solutions proposed for the first objective (to provide a formal semantics for the SPARQL 1.1 Federation Extension) and partially the second objective (to optimise these access using the SPARQL semantics and SPARQL pattern rewriting). Chapter 5 will describe the solutions related to the second objective (to optimise access using existing distributed query processing systems for relational databases) and the solutions for the third objective (to propose a test suite for measuring the performance of the different Distributed SPARQL Query Processing systems) are presented in Chapter 6.

With regard to the first objective, the document presents new advances in the state of the art:

C1.1 A formalisation of the SPARQL 1.1 Federation extension. Currently there are no complete well defined semantics for that part of the SPARQL query language. We provide a formalisation of it, identifying its semantics and problems associated to it.

C1.2 The identification of specific SPARQL optimisation techniques can be considered as a part of the previous contribution. In the formalisation of the SPARQL 1.1 Federation Extension we identified that existing pattern-rewriting rules from SPARQL 1.0 can be applied to the federation extension, allowing us to optimise queries to distributed RDF data sets.

With regard to the second objective, the document presents new advances in the state of the art in the following aspects:

C2.1 We extended the use of distributed query processing systems for its use in querying RDF data using SPARQL. We extended the OGSA-DAI/DQP system for developing the SPARQL-DQP system, a system which uses the WS-DAI specification for using Web services for data access and integration.

C2.2 We also contributed to the state of the art by creating two new data resources, named R2O data resource and the RDF data resource. These data resources provide a Web services layer for accessing and integrating data provided by SPARQL endpoint, RDF databases and data resources wrapped by RDB2RDF systems.

The advances presented with regard to the third objective are the following:

3. OBJECTIVES AND CONTRIBUTIONS

C3.1 We provided a more complete set of queries for evaluating distributed SPARQL query systems. As we will show, the existing distributed SPARQL benchmarks provide a restricted set of queries, emphasising the use of UNION and join operations and not paying much attention to other SPARQL patterns like OPTIONAL. Our contribution provides more queries with the missing SPARQL patterns that allow to improve the evaluation of distributed SPARQL query processing systems.

3.3 Assumptions

The work described in this document is based on the set of assumptions listed below. These assumptions provide a background that facilitate the understanding of the decisions taken for the development of the methodological and technological solutions and the relevance of the contributions presented.

- A1** Users have knowledge of the data stored in the distributed data sets and can issue queries to them. We assume that users know what they are querying and, what data they want to retrieve, so that they can build their queries. We assume that the SPARQL query is given and created by a user.
- A2** The schema for each distributed data set is available to the users. We also assume that users not only have knowledge about the data stored in remote RDF data sets but also have knowledge of the vocabulary used to represent it. Users need to have knowledge about the relations in the remote RDF data set to be able to create a SPARQL query.
- A3** No statistics of any kind are available for the query processing system. We assume that the remote SPARQL endpoint does not provide any statistics about the data it contains, nor SPARQL-DQP has knowledge about the data statistics in the remote RDF data set.
- A4** Data are distributed. We assume that data are stored in several and distributed RDF data sets and for accessing it a single SPARQL query that redirects subqueries to these data sets is needed.

3.4 Hypothesis

Once the assumptions have been identified, the hypotheses of our work are described. These hypotheses cover the main features of the solutions proposed.

- H1** Existing optimisations developed in the context of distributed query processing (mainly for relational databases) can be applied to optimising queries to distributed RDF data. The relational database research community is large and many contributions to the state of the art already exist.

Our hypothesis assumes that this work, or at least part of it, can be used in the context of querying distributed RDF data.

- H2** It is possible to optimise queries to distributed RDF data repositories (using the SPARQL query language) without statistics. The optimisations are based on pattern reordering and at least, these optimisations do not make the query response times worse. These optimisations improve or leave as they are the query response times.
- H3** Existing tools for accessing distributed data sources specifically tailored for large amounts of data are more suitable than existing approaches for querying distributed RDF data. We suppose that the existing work in querying large amounts of data in distributed environments (like life science data used creating for predicting models) is more suitable than existing approaches for querying RDF data. The current amounts of distributed RDF data are too large to be managed by current approaches and focus has to be changed, from one paradigm to another.
- H4** Results are the same before and after applying the rewriting rules. That is, the rewriting rules produce semantically-equivalent SPARQL patterns.

3.5 Restrictions

Finally, there is a set of restrictions that defines the limits of our contribution and establishes future research objectives. These restrictions are the following:

- R1** We only consider the Federation Extension of SPARQL 1.1. We restrict the scope of our research by limiting the parts of SPARQL 1.1 considered. The SPARQL 1.1 contains a main query document and another extension documents which add more functionalities, in addition to those specifying the SPARQL protocol and the query results. We concentrate on the SPARQL 1.1 Federation Extension since it is the most pertinent to our distributed SPARQL query processing research.
- R2** We are not aware of the capabilities or implementation of the remote SPARQL server. We do not know what processing is done by the remote RDF data sets to provide answers to our queries. We only query them, without taking into account internal data representation, remote optimisations or data availability.
- R3** We restrict SPARQL-DQP to not use any list of remote SPARQL endpoint. The remote RDF data sets are to be specified by the user in her query, providing maximum flexibility for querying any remote SPARQL endpoint.

3. OBJECTIVES AND CONTRIBUTIONS

4

Problem formalisation

In this chapter, we give an algebraic formalisation of SPARQL 1.1 including the SPARQL 1.1 federation extension. We restrict ourselves to SPARQL over simple RDF, that is, we disregard higher entailment regimes (see [GO10]) such as RDF(S) or OWL. Our starting point is the existing formalisation of SPARQL described in [PAG09], to which we add the operators SERVICE and BINDINGS proposed in [EP10].

We introduce first the necessary notions about RDF (taken mainly from [PAG09]). Assume there are pairwise disjoint infinite sets I , B , and L (IRIs [DS05], Blank nodes, and Literals, respectively). Then a triple $(s, p, o) \in (I \cup B) \times I \times (I \cup B \cup L)$ is called an *RDF triple*, where s is called the *subject*, p the *predicate* and o the *object*. An *RDF graph* is a set of RDF triples.

Moreover, assume the existence of an infinite set V of variables disjoint from the above sets, and leave UNBOUND to be a reserved symbol that does not belong to any of the previously mentioned sets.

4.1 SPARQL Syntax

The official syntax of SPARQL [PS08] considers operators OPTIONAL, UNION, FILTER, SELECT and concatenation via a point symbol ($.$), to construct graph pattern expressions. Operators SERVICE and BINDINGS are introduced in the SPARQL 1.1 federation extension, the former for allowing users to direct a portion of a query to a particular SPARQL endpoint, and the latter for transferring results that are used to constrain a query. The syntax of the language also considers $\{ \}$ to group patterns, and some implicit rules of precedence and association. In order to avoid ambiguities in the parsing, we follow the approach proposed in [PAG09], and we first present the syntax of SPARQL graph patterns in a more traditional algebraic formalism, using operators AND ($.$), UNION (UNION), OPT (OPTIONAL),

4. PROBLEM FORMALISATION

FILTER (FILTER) and SERVICE (SERVICE), then we introduce the syntax of BINDINGS queries, which use the BINDINGS operator (BINDINGS), and we conclude by defining the syntax of SELECT queries, which use the SELECT operator (SELECT). More precisely, a SPARQL graph pattern expression is defined recursively as follows:

- (1) A tuple from $(I \cup L \cup V) \times (I \cup V) \times (I \cup L \cup V)$ is a graph pattern (a triple pattern).
- (2) If P_1 and P_2 are graph patterns, then expressions $(P_1 \text{ AND } P_2)$, $(P_1 \text{ OPT } P_2)$, and $(P_1 \text{ UNION } P_2)$ are graph patterns.
- (3) If P is a graph pattern and R is a *SPARQL built-in condition*, then the expression $(P \text{ FILTER } R)$ is a graph pattern.
- (4) If P is a graph pattern and $a \in (I \cup V)$, then $(\text{SERVICE } a \ P)$ is a graph pattern.

Moreover, a SPARQL BINDINGS query is defined as follows:

- (5) If P is a graph pattern, S is a nonempty list of pairwise distinct variables and $\{A_1, \dots, A_n\}$ is a nonempty set of lists such that for every $i \in \{1, \dots, n\}$, it holds that A_i and S have the same length and each element in A_i belongs to $(I \cup L \cup \{\text{UNBOUND}\})$, then $(P \text{ BINDINGS } S \ \{A_1, \dots, A_n\})$ is a BINDINGS query.

Finally, assuming that P is either a graph pattern or a BINDINGS query, let $\text{var}(P)$ be the set of variables mentioned in P . Then a SPARQL SELECT query is defined as:

- (6) If P is a graph pattern, and W is a set of variables such that $W \subseteq \text{var}(P)$, then $(\text{SELECT } W \ P)$ is a SELECT query.

It is important to notice that the rules (1)–(3) above were introduced in [PAG09], while we formalise in the rules (4)–(6) the federation extension of SPARQL proposed in [EP10].

In the previous definition, we use the notion of built-in condition for the filter operator. A SPARQL built-in condition is constructed using elements of the set $(I \cup L \cup V)$ and constants, logical connectives (\neg , \wedge , \vee), inequality symbols ($<$, \leq , \geq , $>$), the equality symbol ($=$), unary predicates like `bound`, `isBlank`, and `isIRI`, plus other features (see [PS08] for a complete list). We restrict to the fragment of SPARQL where the built-in condition is a Boolean combination of terms constructed by using $=$ and `bound`, that is: (1) if $?X, ?Y \in V$ and $c \in (I \cup L)$, then `bound(?X)`, `?X = c` and `?X = ?Y` are built-in conditions, and (2) if R_1 and R_2 are built-in conditions, then $(\neg R_1)$, $(R_1 \vee R_2)$ and $(R_1 \wedge R_2)$ are built-in conditions. It should be noticed that the results of the thesis can be easily extended to the other built-in predicates in SPARQL.

Let P be either a graph pattern or a BINDINGS query or a SELECT query. In the rest of the thesis, we use $\text{var}(P)$ to denote the set of variables occurring in P . Similarly, for a built-in condition R , we use $\text{var}(R)$ to denote the set of variables occurring in R .

4.2 SPARQL Semantics

To define the semantics of SPARQL queries, we need to introduce some extra terminology from [PAG09]. A mapping μ from V to $(I \cup B \cup L)$ is a partial function $\mu : V \rightarrow (I \cup B \cup L)$. Abusing notation, for a triple pattern t we denote by $\mu(t)$ the triple obtained by replacing the variables in t according to μ . The domain of μ , denoted by $\text{dom}(\mu)$, is the subset of V where μ is defined. Two mappings μ_1 and μ_2 are compatible when for all $?X \in \text{dom}(\mu_1) \cap \text{dom}(\mu_2)$, it is the case that $\mu_1(?X) = \mu_2(?X)$, i.e. when $\mu_1 \cup \mu_2$ is also a mapping.

Let Ω_1 and Ω_2 be sets of mappings. Then the join of, the union of, the difference between and the left outer-join between Ω_1 and Ω_2 are defined as follows [PAG09]:

$$\begin{aligned} \Omega_1 \bowtie \Omega_2 &= \{\mu_1 \cup \mu_2 \mid \mu_1 \in \Omega_1, \mu_2 \in \Omega_2 \text{ and } \mu_1, \mu_2 \text{ are compatible mappings}\}, \\ \Omega_1 \cup \Omega_2 &= \{\mu \mid \mu \in \Omega_1 \text{ or } \mu \in \Omega_2\}, \\ \Omega_1 \setminus \Omega_2 &= \{\mu \in \Omega_1 \mid \text{for all } \mu' \in \Omega_2, \mu \text{ and } \mu' \text{ are not compatible}\}, \\ \Omega_1 \bowtie \Omega_2 &= (\Omega_1 \bowtie \Omega_2) \cup (\Omega_1 \setminus \Omega_2). \end{aligned}$$

Next we use the preceding operators to give semantics to graph pattern expressions, BINDINGS queries and SELECT queries. More specifically, we define this semantics as a function $\llbracket \cdot \rrbracket_G^{DS}$, which takes as input any of these types of queries and returns a set of mappings. In this definition, we assume given a partial function ep from the set I of IRIs such that for every $c \in I$, if $\text{ep}(c)$ is defined, then $\text{ep}(c)$ is an RDF graph. Intuitively, function ep is defined for an element $c \in I$ ($c \in \text{dom}(\text{ep})$) if and only if c is the IRI of a SPARQL endpoint, and $\text{ep}(c)$ is the default RDF graph of that endpoint¹. Moreover, in this definition μ_\emptyset represents the mapping with empty domain (which is compatible with any other mapping).

The evaluation of a graph pattern P over an RDF graph G , denoted by $\llbracket P \rrbracket_G^{DS}$, is defined recursively as follows:

- (1) If P is a triple pattern t , then $\llbracket P \rrbracket_G^{DS} = \{\mu \mid \text{dom}(\mu) = \text{var}(t) \text{ and } \mu(t) \in G\}$.
- (2) If P is $(P_1 \text{ AND } P_2)$, then $\llbracket P \rrbracket_G^{DS} = \llbracket P_1 \rrbracket_G^{DS} \bowtie \llbracket P_2 \rrbracket_G^{DS}$.
- (3) If P is $(P_1 \text{ OPT } P_2)$, then $\llbracket P \rrbracket_G^{DS} = \llbracket P_1 \rrbracket_G^{DS} \bowtie \llbracket P_2 \rrbracket_G^{DS}$.

¹For simplicity, we only assume a single (default) graph and no named graphs per remote SPARQL endpoint.

4. PROBLEM FORMALISATION

(4) If P is $(P_1 \text{ UNION } P_2)$, then $\llbracket P \rrbracket_G^{DS} = \llbracket P_1 \rrbracket_G^{DS} \cup \llbracket P_2 \rrbracket_G^{DS}$.

(5) If P is $(\text{SERVICE } c \ P_1)$ with $c \in I$, then

$$\llbracket P \rrbracket_G^{DS} = \begin{cases} \llbracket P_1 \rrbracket_{\text{ep}(c)} & \text{if } c \in \text{dom}(\text{ep}) \\ \{\mu_\emptyset\} & \text{otherwise} \end{cases}$$

(6) If P is $(\text{SERVICE } ?X \ P_1)$ with $?X \in V$, then $\llbracket P \rrbracket_G^{DS}$ is equal to:

$$\bigcup_{c \in I} \left\{ \mu \mid \text{there exists } \mu' \in \llbracket (\text{SERVICE } c \ P_1) \rrbracket_G^{DS} \text{ s.t. } \text{dom}(\mu) = (\text{dom}(\mu') \cup \{?X\}), \right. \\ \left. \mu(?X) = c \text{ and } \mu(?Y) = \mu'(?Y) \text{ for every } ?Y \in \text{dom}(\mu') \right\}$$

Next, present the semantics of the FILTER operator proposed in [PAG09]. Given a mapping μ and a built-in condition R , μ is said to satisfy R , denoted by $\mu \models R$, if:

1. R is $\text{bound}(?X)$ and $?X \in \text{dom}(\mu)$;
2. R is $?X = c$, $?X \in \text{dom}(\mu)$ and $\mu(?X) = c$;
3. R is $?X = ?Y$, $?X \in \text{dom}(\mu)$, $?Y \in \text{dom}(\mu)$ and $\mu(?X) = \mu(?Y)$.
4. R is $(\neg R_1)$, R_1 is a built-in condition, and it is not the case that $\mu \models R_1$;
5. R is $(R_1 \vee R_2)$, R_1 and R_2 are built-in conditions, and $\mu \models R_1$ or $\mu \models R_2$;
6. R is $(R_1 \wedge R_2)$, R_1 and R_2 are built-in conditions, $\mu \models R_1$ and $\mu \models R_2$;

Then given an RDF graph G and a graph pattern $P = (P_1 \text{ FILTER } R)$, the evaluation of P over G , denoted by $\llbracket P \rrbracket_G^{DS}$, is defined as $\{\mu \in \llbracket P_1 \rrbracket_G^{DS} \mid \mu \models R\}$.

Moreover, the semantics of BINDINGS queries is defined as follows. Given a list $S = [?X_1, \dots, ?X_\ell]$ of pairwise distinct variables, where $\ell \geq 1$, and a list $A = [a_1, \dots, a_\ell]$ of values from $(I \cup L \cup \{\text{UNBOUND}\})$, let $\mu_{S,A}$ be a mapping with domain $\{?X_i \mid i \in \{1, \dots, \ell\} \text{ and } a_i \in (I \cup L)\}$ and such that $\mu_{S,A}(?X_i) = a_i$ for every $?X_i \in \text{dom}(\mu_{S,A})$. Then

(7) If $P = (P_1 \text{ BINDINGS } S \ \{A_1, \dots, A_n\})$ is a BINDINGS query:

$$\llbracket P \rrbracket_G^{DS} = \llbracket P_1 \rrbracket_G^{DS} \bowtie \{\mu_{S,A_1}, \dots, \mu_{S,A_n}\}.$$

Finally, the semantics of SELECT queries is defined as follows. Given a mapping $\mu : V \rightarrow (I \cup B \cup L)$ and a set of variables $W \subseteq V$, the restriction of μ to W , denoted by $\mu|_W$, is a mapping such that $\text{dom}(\mu|_W) = (\text{dom}(\mu) \cap W)$ and $\mu|_W(?X) = \mu(?X)$ for every $?X \in (\text{dom}(\mu) \cap W)$. Then

(8) If $P = (\text{SELECT } W \ P_1)$ is a SELECT query: $\llbracket P \rrbracket_G^{DS} = \{\mu|_W \mid \mu \in \llbracket P_1 \rrbracket_G^{DS}\}$.

It is important to notice that the rules (1)–(4) above were introduced in [PAG09], while we propose in the rules (5)–(8) a semantics for the operators SERVICE and BINDINGS introduced in [EP10]. Intuitively, if $c \in I$ is the IRI of a SPARQL endpoint, then the idea behind the definition of $(\text{SERVICE } c \ P_1)$ is to evaluate query P_1 in the SPARQL endpoint specified by c . On the other hand, if $c \in I$ is not the IRI of a SPARQL endpoint, then $(\text{SERVICE } c \ P_1)$ leaves unbounded all the variables in P_1 , as this query cannot be evaluated in this case. This idea is formalised by making μ_\emptyset the only mapping in the evaluation of $(\text{SERVICE } c \ P_1)$ if $c \notin \text{dom}(\text{ep})$. In the same way, $(\text{SERVICE } ?X \ P_1)$ is defined by considering all the possible IRIs for the variable $?X$, that is, all the values $c \in I$. In fact, $(\text{SERVICE } ?X \ P_1)$ is defined as the union of the evaluation of the graph patterns $(\text{SERVICE } c \ P_1)$ for the values $c \in I$, but also storing in $?X$ the IRIs from where the values of the variables in P_1 are coming from. Finally, the idea behind the definition of $(P_1 \ \text{BINDINGS } S \ \{A_1, \dots, A_n\})$ is to constrain the values of the variables in S to the values specified in A_1, \dots, A_n .

Example 1. Assume that G is an RDF graph that uses triples of the form $(a, \text{service_address}, b)$ to indicate that a SPARQL endpoint with name a is located at the IRI b . Moreover, let P be the following SPARQL query:

$$\left[\text{SELECT } \{?X, ?N, ?E\} \right. \\ \left. \left(\left((?X, \text{service_address}, ?Y) \text{ AND } (\text{SERVICE } ?Y \ (?N, \text{email}, ?E)) \right) \right. \right. \\ \left. \left. \text{BINDINGS } [?N] \{[John], [Peter]\} \right) \right]$$

Query P is used to compute the list of names and email addresses that can be retrieved from the SPARQL endpoints stored in an RDF graph. In fact, if $\mu \in \llbracket P \rrbracket_G^{DS}$, then $\mu(?X)$ is the name of a SPARQL endpoint stored in G , $\mu(?N)$ is the name of a person stored in that SPARQL endpoint and $\mu(?E)$ is the email address of that person. Moreover, the operator BINDINGS in this query is used to filter the values of the variable $?N$. Specifically, if $\mu \in \llbracket P \rrbracket_G^{DS}$, then $\mu(?N)$ is either John or Peter. \square

The goal of the rules (5)–(8) is to define in an unambiguous way what the result of evaluating an expression containing the operators SERVICE and BINDINGS should be. As such, these rules should not be considered as an implementation of the language. In fact, a direct implementation of the rule (6), that defines the semantics of a pattern of the form $(\text{SERVICE } ?X \ P_1)$, would involve evaluating a particular query in every possible SPARQL endpoint, which is obviously unfeasible in practice. In the next section, we face this issue and, in particular, we introduce a syntactic condition on SPARQL queries that ensures that a pattern of the form $(\text{SERVICE } ?X \ P_1)$ can be evaluated by only considering

4. PROBLEM FORMALISATION

a finite set of SPARQL endpoints, whose IRIs are actually taken from the RDF graph where the query is being evaluated.

We conclude this section by presenting the semantics of the FILTER operator proposed in [PAG09]. Given a mapping μ and a built-in condition R , we say that μ satisfies R , denoted by $\mu \models R$, if (omitting the usual rules for Boolean connectives):

1. R is $\text{bound}(\text{?}X)$ and $\text{?}X \in \text{dom}(\mu)$;
2. R is $\text{?}X = c$, $\text{?}X \in \text{dom}(\mu)$ and $\mu(\text{?}X) = c$;
3. R is $\text{?}X = \text{?}Y$, $\text{?}X \in \text{dom}(\mu)$, $\text{?}Y \in \text{dom}(\mu)$ and $\mu(\text{?}X) = \mu(\text{?}Y)$.
4. R is $(\neg R_1)$, R_1 is a built-in condition, and it is not the case that $\mu \models R_1$;
5. R is $(R_1 \vee R_2)$, R_1 and R_2 are built-in conditions, and $\mu \models R_1$ or $\mu \models R_2$;
6. R is $(R_1 \wedge R_2)$, R_1 and R_2 are built-in conditions, $\mu \models R_1$ and $\mu \models R_2$;

Then given an RDF graph G and a graph pattern $P = (P_1 \text{ FILTER } R)$, the evaluation of P over G , denoted by $\llbracket P \rrbracket_G^{DS}$, is defined as $\{\mu \in \llbracket P_1 \rrbracket_G^{DS} \mid \mu \models R\}$.

4.3 On Evaluating the SERVICE Operator in SPARQL queries

As we pointed out in the previous section, the evaluation of a pattern of the form $(\text{SERVICE } \text{?}X \ P)$ is unfeasible unless the variable $\text{?}X$ is bound to a finite set of IRIs. This notion of *boundedness* is one of the most significant and unclear concepts in the SPARQL federation extension. In fact, the current version of the specification [EP10] only specifies that a variable $\text{?}X$ in a pattern of the form $(\text{SERVICE } \text{?}X \ P)$ *must be bound*, but without providing a formal definition of what that means. Here we provide a formalisation of this concept, studying the complexity issues associated with it.

4.3.1 The notion of boundedness

In Example 1, we present a SPARQL query containing a pattern $(\text{SERVICE } \text{?}Y \ (\text{?}N, \text{email}, \text{?}E))$. Given that variable $\text{?}Y$ is used to store the address of a remote SPARQL endpoint to be queried, it is important to assign a value to $\text{?}Y$ prior to the evaluation of the SERVICE pattern. In the case of the query in Example 1, this needs of a simple strategy: given an RDF graph G , first compute $\llbracket (\text{?}X, \text{service_address}, \text{?}Y) \rrbracket_G^{DS}$, and then for every μ in this set, compute $\llbracket (\text{SERVICE } a \ (\text{?}N, \text{email}, \text{?}E)) \rrbracket_G^{DS}$ with $a = \mu(\text{?}Y)$. More generally, SPARQL pattern $(\text{SERVICE } \text{?}Y \ (\text{?}N, \text{email}, \text{?}E))$ can be evaluated

in this case as only a finite set of values from the domain of G need to be considered as the possible values of $?Y$. This idea naturally gives rise to the following notion of boundedness for the variables of a SPARQL query. In the definition of this notion, $\text{dom}(G)$ refers to the domain of G , that is, the set of elements from $(I \cup B \cup L)$ that are mentioned in G , and $\text{dom}(P)$ refers to the set of elements from $(I \cup L)$ that are mentioned in P .

Definition 1 (Boundedness). *Let P be a SPARQL query and $?X \in \text{var}(P)$. Then $?X$ is bound in P if one of the following conditions holds:*

- *P is a graph pattern, and for every RDF graph G and mapping $\mu \in \llbracket P \rrbracket_G^{DS}$, it holds that $?X \in \text{dom}(\mu)$ and $\mu(?X) \in (\text{dom}(G) \cup \text{dom}(P))$.*
- *P is a SELECT query (SELECT W P_1) and $?X$ is bound in P_1 .*

The BINDINGS operator can make a variable $?X$ in a query P to be bound by assigning to it a fixed set of values. Given that these values are not necessarily mentioned in the RDF graph G where P is being evaluated, the previous definition first imposes the condition that $?X \in \text{dom}(\mu)$, and then not only considers the case $\mu(?X) \in \text{dom}(G)$ but also the case $\mu(?X) \in \text{dom}(P)$. As an example of the above definition, we note that variable $?Y$ is bound in the graph pattern

$$P_1 = ((?X, \text{service_address}, ?Y) \text{ AND } (\text{SERVICE } ?Y (?N, \text{email}, ?E))),$$

as for every RDF graph G and mapping $\mu \in \llbracket P_1 \rrbracket_G^{DS}$, we know that $?Y \in \text{dom}(\mu)$ and $\mu(?Y) \in \text{dom}(G)$. Moreover, we also have that variable $?Y$ is bound in (SELECT $\{?X, ?N, ?E\}$ P_1) as $?Y$ is bound in graph pattern P_1 .

A natural way to ensure that a SPARQL query P can be evaluated in practice is by imposing the restriction that for every sub-pattern (SERVICE $?X$ P_1) of P , it holds that $?X$ is bound in P . However, in the following theorem we show that such a condition is undecidable and, thus, a SPARQL query engine would not be able to check it in order to ensure that a query can be evaluated.

Theorem 1. *The problem of verifying, given a SPARQL query P and a variable $?X \in \text{var}(P)$, whether $?X$ is bound in P is undecidable.*

The fact that the notion of boundedness is undecidable prevents one from using it as a restriction over the variables in SPARQL queries. To overcome this limitation, we introduce here a syntactic condition that ensures that a variable is bound in a pattern and that can be efficiently verified.

Definition 2 (Strong boundedness). *Let P be a SPARQL query. Then the set of strongly bound variables in P , denoted by $\text{SB}(P)$, is recursively defined as follows:*

- *if $P = t$, where t is a triple pattern, then $\text{SB}(P) = \text{var}(t)$;*

4. PROBLEM FORMALISATION

- if $P = (P_1 \text{ AND } P_2)$, then $\text{SB}(P) = \text{SB}(P_1) \cup \text{SB}(P_2)$;
- if $P = (P_1 \text{ UNION } P_2)$, then $\text{SB}(P) = \text{SB}(P_1) \cap \text{SB}(P_2)$;
- if $P = (P_1 \text{ OPT } P_2)$ or $P = (P_1 \text{ FILTER } R)$, then $\text{SB}(P) = \text{SB}(P_1)$;
- if $P = (\text{SERVICE } c \ P_1)$, with $c \in I$, or $P = (\text{SERVICE } ?X \ P_1)$, with $?X \in V$, then $\text{SB}(P) = \emptyset$;
- if $P = (P_1 \text{ BINDINGS } S \{A_1, \dots, A_n\})$, then $\text{SB}(P) = \text{SB}(P_1) \cup \{?X \mid ?X \text{ is in } S \text{ and for every } i \in \{1, \dots, n\}, \text{ it holds that } ?X \in \text{dom}(\mu_{S, A_i})\}$.
- if $P = (\text{SELECT } W \ P_1)$, then $\text{SB}(P) = (W \cap \text{SB}(P_1))$.

The previous definition recursively collects from a SPARQL query P a set of variables that are guaranteed to be bound in P . For example, if P is a triple pattern t , then $\text{SB}(P) = \text{var}(t)$ as one knows that for every variable $?X \in \text{var}(t)$ and for every RDF graph G , if $\mu \in \llbracket t \rrbracket_G^{DS}$, then $?X \in \text{dom}(\mu)$ and $\mu(?X) \in \text{dom}(G)$. In the same way, if $P = (P_1 \text{ AND } P_2)$, then $\text{SB}(P) = \text{SB}(P_1) \cup \text{SB}(P_2)$ as one knows that if $?X$ is bound in P_1 or in P_2 , then $?X$ is bound in P . As a final example, notice that if $P = (P_1 \text{ BINDINGS } S \{A_1, \dots, A_n\})$ and $?X$ is a variable mentioned in S such that $?X \in \text{dom}(\mu_{S, A_i})$ for every $i \in \{1, \dots, n\}$, then $?X \in \text{SB}(P)$. In this case, one knows that $?X$ is bound in P since $\llbracket P \rrbracket_G^{DS} = \llbracket P_1 \rrbracket_G^{DS} \bowtie \{\mu_{S, A_1}, \dots, \mu_{S, A_n}\}$ and $?X$ is in the domain of each one of the mappings μ_{S, A_i} , which implies that $\mu(?X) \in \text{dom}(P)$ for every $\mu \in \llbracket P \rrbracket_G^{DS}$. In the following proposition, we formally show that our intuition about $\text{SB}(P)$ is correct, in the sense that every variable in this set is bound in P .

Proposition 1. *For every SPARQL query P and variable $?X \in \text{var}(P)$, if $?X \in \text{SB}(P)$, then $?X$ is bound in P .*

Given a SPARQL query P and a variable $?X \in \text{var}(P)$, it can be efficiently verified whether $?X$ is strongly bound in P . Thus, a natural and efficiently verifiable way to ensure that a SPARQL query P can be evaluated in practice is by imposing the restriction that for every sub-pattern $(\text{SERVICE } ?X \ P_1)$ of P , it holds that $?X$ is strongly bound in P . However, this notion still needs to be modified in order to be useful in practice, as shown by the following examples.

Example 2. *Assume first that P_1 is the following graph pattern:*

$$P_1 = ((?X, \text{service_description}, ?Z) \text{ UNION } ((?X, \text{service_address}, ?Y) \text{ AND } (\text{SERVICE } ?Y (?N, \text{email}, ?E)))).$$

That is, either $?X$ and $?Z$ store the name of a SPARQL endpoint and a description of its functionalities, or $?X$ and $?Y$ store the name of a SPARQL endpoint and the IRI where it is located

(together with a list of names and email addresses retrieved from that location). Variable $?Y$ is neither bound nor strongly bound in P_1 . However, there is a simple strategy that ensures that P_1 can be evaluated over an RDF graph G : first compute $\llbracket (?X, \text{service_description}, ?Z) \rrbracket_G^{DS}$, then compute $\llbracket (?X, \text{service_address}, ?Y) \rrbracket_G^{DS}$, and finally for every μ in the set $\llbracket (?X, \text{service_address}, ?Y) \rrbracket_G^{DS}$, compute $\llbracket (\text{SERVICE } a (?N, \text{email}, ?E)) \rrbracket_G^{DS}$ with $a = \mu(?Y)$. In fact, the reason why P_1 can be evaluated in this case is that $?Y$ is bound (and strongly bound) in the sub-pattern $((?X, \text{service_address}, ?Y) \text{ AND } (\text{SERVICE } ?Y (?N, \text{email}, ?E)))$ of P_1 .

As a second example, assume that G is an RDF graph that uses triples of the form $(a_1, \text{related_with}, a_2)$ to indicate that the SPARQL endpoints located at the IRIs a_1 and a_2 store related data. Moreover, assume that P_2 is the following graph pattern:

Example 3.

$$P_2 = ((?U_1, \text{related_with}, ?U_2) \text{ AND } (\text{SERVICE } ?U_1 ((?N, \text{email}, ?E) \text{ OPT } (\text{SERVICE } ?U_2 (?N, \text{phone}, ?F)))).$$

When this query is evaluated over the RDF graph G , it returns for every tuple $(a_1, \text{related_with}, a_2)$ in G , the list of names and email addresses that that can be retrieved from the SPARQL endpoint located at a_1 , together with the phone number for each person in this list for which this data can be retrieved from the SPARQL endpoint located at a_2 (recall that graph pattern $(\text{SERVICE } ?U_2 (?N, \text{phone}, ?F))$ is nested inside the first SERVICE operator in P_2). To evaluate this query over an RDF graph, first it is necessary to determine the possible values for variable $?U_1$, and then to submit the query

$$((?N, \text{email}, ?E) \text{ OPT } (\text{SERVICE } ?U_2 (?N, \text{phone}, ?F)))$$

to each one of the endpoints located at the IRIs stored in $?U_1$. In this case, variable $?U_2$ is bound (and also strongly bound) in P_2 . However, this variable is not bound in the graph pattern

$$((?N, \text{email}, ?E) \text{ OPT } (\text{SERVICE } ?U_2 (?N, \text{phone}, ?F)))$$

, which has to be evaluated in some of the SPARQL endpoints stored in the RDF graph where P_2 is being evaluated, something that is unfeasible in practice. Notice that the difficulties in evaluating P_2 are caused by the nesting of SERVICE operators (more precisely, by the fact that P_2 has a sub-pattern of the form $(\text{SERVICE } ?X_1 Q_1)$, where Q_1 has in turn a sub-pattern of the form $(\text{SERVICE } ?X_2 Q_2)$ such that $?X_2$ is bound in P_2 but not in Q_1). \square

In the following section, we use the concept of strongly boundedness to define a notion that ensures that a SPARQL query containing the SERVICE operator can be evaluated in practice, and which takes into consideration the ideas presented in Example 2.

4. PROBLEM FORMALISATION

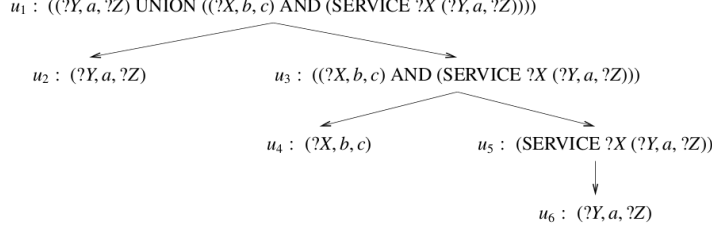


Figure 4.1: Parse tree $\mathcal{T}(Q)$ for the graph pattern $Q = ((?Y, a, ?Z) \text{ UNION } ((?X, b, c) \text{ AND } (\text{SERVICE } ?X (?Y, a, ?Z))))$.

4.3.2 The notion of service-safeness: Considering sub-patterns and nested SERVICE operators

The goal of this section is to provide a condition that ensures that a SPARQL query containing the SERVICE operator can be safely evaluated. To this end, we first need to introduce some terminology. Given a SPARQL query P , define $\mathcal{T}(P)$ as the *parse tree* of P . In this tree, every node corresponds to a sub-pattern of P . An example of a parse tree of a pattern Q is shown in Figure 4.1. In this figure, $u_1, u_2, u_3, u_4, u_5, u_6$ are the identifiers of the nodes of the tree, which are labelled with the sub-patterns of Q . It is important to notice that in this tree we do not make any distinction between the different operators in SPARQL, we just store the structure of the sub-patterns of a SPARQL query.

Tree $\mathcal{T}(P)$ is used to define the notion of service-boundedness, which extends the concept of boundedness, introduced in the previous section, to consider variables that are bound inside sub-patterns and nested SERVICE operators. It should be noticed that these two features were identified in the previous section as important for the definition of a notion of boundedness (see Example 2).

Definition 3 (Service-boundedness). *A SPARQL query P is service-bound if for every node u of $\mathcal{T}(P)$ with label $(\text{SERVICE } ?X P_1)$, it holds that: (1) there exists a node v of $\mathcal{T}(P)$ with label P_2 such that v is an ancestor of u in $\mathcal{T}(P)$ and $?X$ is bound in P_2 ; (2) P_1 is service-bound.*

For example, query Q in Figure 4.1 is service-bound. In fact, condition (1) of Definition 3 is satisfied as u_5 is the only node in $\mathcal{T}(Q)$ having as label a SERVICE graph pattern, in this case $(\text{SERVICE } ?X (?Y, a, ?Z))$, and for the node u_3 , it holds that: u_3 is an ancestor of u_5 in $\mathcal{T}(P)$, the label of u_3 is $P = ((?X, b, c) \text{ AND } (\text{SERVICE } ?X (?Y, a, ?Z)))$ and $?X$ is bound in P . Moreover, condition (2) of Definition 3 is satisfied as the sub-pattern $(?Y, a, ?Z)$ of the label of u_5 is also service-bound.

The notion of service-boundedness captures our intuition about the condition that a SPARQL query containing the SERVICE operator should satisfy. Unfortunately, the following theorem shows that such a condition is undecidable and, thus, a query engine would not be able to check it in order to ensure that a query can be evaluated.

Theorem 2. *The problem of verifying, given a SPARQL query P , whether P is service-bound is undecidable.*

As for the case of the notion of boundedness, the fact that the notion of service-boundedness is undecidable prevents one from using it as a restriction over the variables used in SERVICE calls. To overcome this limitation, we replace the restriction that the variables used in SERVICE calls are bound by the decidable restriction that they are strongly bound. In this way, we obtain a syntactic condition over SPARQL patterns that ensures that they are service-bound, and which can be efficiently verified.

Definition 4 (Service-safeness). *A SPARQL query P is service-safe if for every node u of $\mathcal{T}(P)$ with label (SERVICE ?X P_1), it holds that: (1) there exists a node v of $\mathcal{T}(P)$ with label P_2 such that v is an ancestor of u in $\mathcal{T}(P)$ and ?X \in SB(P_2); (2) P_1 is service-safe.*

Proposition 2. *If a SPARQL query P is service-safe, then P is service-bound.*

The notion of service-safeness is used in our system to verify that a SPARQL query can be evaluated in practice. More precisely, our system uses a bottom-up algorithm over the parse tree $\mathcal{T}(Q)$ of a SPARQL query Q for validating the service-safeness condition. This procedure traverses the parse tree $\mathcal{T}(Q)$ twice for assuring that Q can be correctly evaluated. In the first traversal, for each node identifier u of $\mathcal{T}(Q)$, the algorithm computes the set of strongly bound variables for the label P of u . For example, in the parse tree shown in Figure 4.1, the variable ?X is identified as the only strongly bound variable for the label of the node with identifier u_3 . In the second traversal, the bottom-up algorithm uses these sets of strongly bound variables to check whether for every node identifier u of $\mathcal{T}(Q)$ with label of the form (SERVICE ?X P), there exists a node v of $\mathcal{T}(Q)$ with label P' such that v is an ancestor of u in $\mathcal{T}(Q)$ and ?X is strongly bound in P' . If this second condition is fulfilled, then the algorithm returns *true* to indicate that Q is service-safe. Otherwise, the procedure returns *no*.

4.4 Static Optimisations

If a SPARQL query Q including the SERVICE operator has to be evaluated in a SPARQL endpoint A , then some of the sub-queries of Q may have to be evaluated in some external SPARQL endpoints. Thus, the problem of optimising the evaluation of Q in A , and, in particular, the problem of reordering Q in A to optimise this evaluation, becomes particularly relevant in this scenario, as in some cases one cannot rely on the optimises of the external SPARQL endpoints. Motivating by this, we present in this section some optimisation techniques that extend the techniques presented in [PAG09] to the case of SPARQL queries using the SERVICE operator, and which can be applied to a considerable number of SPARQL federated queries.

4. PROBLEM FORMALISATION

4.4.1 Optimisation via well-designed patterns

In [PAG09, MSL10], the authors study the complexity of evaluating the fragment of SPARQL consisting of the operators AND, UNION, OPT and FILTER. One of the conclusions of these papers is that the main source of complexity in SPARQL comes from the use of the OPT operator. In fact, it is proved in [PAG09] that the complexity of the problem of verifying, given a mapping μ , a SPARQL pattern P and an RDF graph G , whether $\mu \in \llbracket P \rrbracket_G^{DS}$ is PSPACE-complete, and it is proved in [MSL10] that this bound remains the same if only the OPT operator is allowed in SPARQL patterns. In light of these results, it was introduced in [PAG09] a fragment of SPARQL that forbids a special form of interaction between variables appearing in optional parts, which rarely occurs in practice. The patterns in this fragment, which are called well-designed patterns [PAG09], can be evaluated more efficiently and are suitable for reordering and optimisation. In this section, we extend the definition of the notion of being well-designed to the case of SPARQL patterns using the SERVICE operator, and prove that the reordering rules proposed in [PAG09], for optimising the evaluation of well-designed patterns, also hold in this extension. The use of these rules allows to reduce the number of tuples being transferred and joined in federated queries, and hence our implementation benefits from this as shown in Chapter 5.

Let P be a graph pattern constructed by using the operators AND, OPT, FILTER and SERVICE, and assume that P satisfies the safety condition that for every sub-pattern $(P_1 \text{ FILTER } R)$ of P , it holds that $\text{var}(R) \subseteq \text{var}(P_1)$. Then, by following [PAG09], we say that P is well-designed if for every sub-pattern $P' = (P_1 \text{ OPT } P_2)$ of P and for every variable $?X$ occurring in P : If $?X$ occurs both inside P_2 and outside P' , then it also occurs in P_1 . All the graph patterns given in the previous sections are well-designed. On the other hand, the following pattern P is not well-designed:

$$((?X, \text{nickname}, ?Y) \text{ AND } (\text{SERVICE } c ((?X, \text{email}, ?U) \text{ OPT } (?Y, \text{email}, ?V))))),$$

as for the sub-pattern $P' = (P_1 \text{ OPT } P_2)$ of P with $P_1 = (?X, \text{email}, ?U)$ and $P_2 = (?Y, \text{email}, ?V)$, we have that $?Y$ occurs in P_2 and outside P' in the triple pattern $(?X, \text{nickname}, ?Y)$, but it does not occur in P_1 . Given an RDF graph G , graph pattern P retrieves from G a list of people with their nicknames, and retrieves from the SPARQL endpoint located at the IRI c the email addresses of these people and, optionally, the email addresses associated to their nicknames. What is unnatural about this graph pattern is the fact that $(?Y, \text{email}, ?V)$ is giving optional information for $(?X, \text{nickname}, ?Y)$, but in P appears as giving optional information for $(?X, \text{email}, ?U)$. In fact, it could happen that some of the results retrieved by using the triple pattern $(?X, \text{nickname}, ?Y)$ are not included in the final answer of P , as the value of variable $?Y$ in these intermediate results could be incompatible with the values

for this variable retrieved by using the triple pattern $(?Y, \text{email}, ?V)$. To overcome this limitation, one should use instead the following well-designed SPARQL graph pattern:

$$(((?X, \text{nickname}, ?Y) \text{ AND } (\text{SERVICE } c (?X, \text{email}, ?U))) \text{ OPT } (\text{SERVICE } c (?Y, \text{email}, ?V))).$$

In the following proposition, we show that well-designed patterns including the SERVICE operator are suitable for reordering and, thus, for optimisation.

Proposition 3. *Let P be a well-designed pattern and P' a pattern obtained from P by using one of the following reordering rules:*

$$\begin{aligned} ((P_1 \text{ OPT } P_2) \text{ FILTER } R) &\longrightarrow ((P_1 \text{ FILTER } R) \text{ OPT } P_2), \\ (P_1 \text{ AND } (P_2 \text{ OPT } P_3)) &\longrightarrow ((P_1 \text{ AND } P_2) \text{ OPT } P_3), \\ ((P_1 \text{ OPT } P_2) \text{ AND } P_3) &\longrightarrow ((P_1 \text{ AND } P_3) \text{ OPT } P_2). \end{aligned}$$

Then P' is a well-designed pattern equivalent to P .

In order to demonstrate that well-designed patterns including the SERVICE operator can also be rewritten preserving the semantics of the original pattern we have to refer to some terminology already introduced in [PAG09]. We say that a pattern P' is a *reduction* of a pattern P , if P' can be obtained from P by replacing a sub-formula $(P_1 \text{ OPT } P_2)$ of P by P_1 , that is, if P' is obtained by deleting some optional part of P . The reflexive and transitive closure of the reduction relation is denoted by \sqsubseteq . Thus, for example, if $P'' = (t_1 \text{ AND } t_2)$, then $P'' \sqsubseteq P$ since P'' is a reduction of P' and P' is a reduction of P . We note that if $P' \sqsubseteq P$ and P is well designed, then P' is well designed. If $P' \sqsubseteq P$ and $P' \neq P$, then we say that P' is a proper reduction of P , denoted by $P' \triangleleft P$.

Next, to show that the application of these rules does not affect the evaluation of the SPARQL pattern, we extend the proof in [PAG09] which shows the same result. In order to extend this proof, we need to demonstrate first the following claim but for patterns containing the SERVICE operator. Demonstrating this claim we show that every variable $?X \in \text{var}(Q)$ will have a mapping $\mu \in \llbracket Q \rrbracket_{ep(a)}$, for every reduction of that pattern Q . Which is necessary condition for proving that the evaluation of the patterns modified using the rewriting rules are the same.

Claim 1. *Given a dataset D , a well-designed pattern Q , (not necessarily well-designed), and a variable $?X \in \text{var}(Q)$, if for every pattern $O \sqsubseteq Q$ we have that $?X \in \text{var}(O)$, then $?X \in \text{dom}(\mu)$ for every $\mu \in \llbracket Q \rrbracket_G^{DS}$.*

The proof of the Theorems, Claims and Propositions in this Chapter can be found in Appendix A.1.

4. PROBLEM FORMALISATION

In this chapter we provided a formalisation for the SPARQL 1.1 Federation Extension. We extended the work in [PAG09], by adding the new operators described in the W3C document, namely SERVICE and BINDINGS. This helped us to understand and identify the problems of the federation extension, which mainly concentrate in the ambiguity of the SERVICE VAR pattern description. We identified the origin of these ambiguities and proposed a way for correctly evaluating such patterns in a SPARQL query.

Finally, we proposed a set of optimisation's based on the rewriting rules identified in [PAG09]. These rules were initially identified for SPARQL 1.0 query patterns. We proved that these rules can also be applied in a distributed scenario and adapted them to it.

5

Accessing Distributed RDF data stores

In this chapter we describe how we access distributed RDF data sets, both using single direct queries to the data sources as well as in a federated manner. Firstly we introduce what RDF data stores we access (RDB2RDF systems, RDF databases and SPARQL endpoints). Next, we describe how we developed SPARQL-DQP, our solution for accessing distributed RDF datasets. SPARQL-DQP extends the OGSA-DQP [ea09] for accessing distributed RDF datasets implementing the SPARQL 1.1 Federation Extension.

5.1 Introduction to Web service data access

The WS-DAI [AKP⁺06] specification defines interfaces to access data resources as web services, and is a proposed recommendation of the Open Grid Forum¹. The general WS-DAI specification has two extended realisations, one for accessing relational databases (WS-DAIR) and another for accessing XML databases (WS-DAIX) and work is being done to provide another extended realisation for RDF data (WS-DAI-RDF).

The key elements of the WS-DAI specification are data services. A data service is a Web service that implements one or more of the DAIS-WG specified interfaces to provide access to data resources. A data resource is a source of data (relational databases, XML databases, file systems). The WS-DAI specification defines concepts, properties and messages that can be used in the definition of data services. It does not provide elements to manage these data sources (i.e. means for managing DBMS), only for accessing them. Data resources are divided in externally managed data resources (the data is stored using an existing data management system) and service managed data resource (it does not normally

¹<http://www.gridforum.org/>

5. ACCESSING DISTRIBUTED RDF DATA STORES

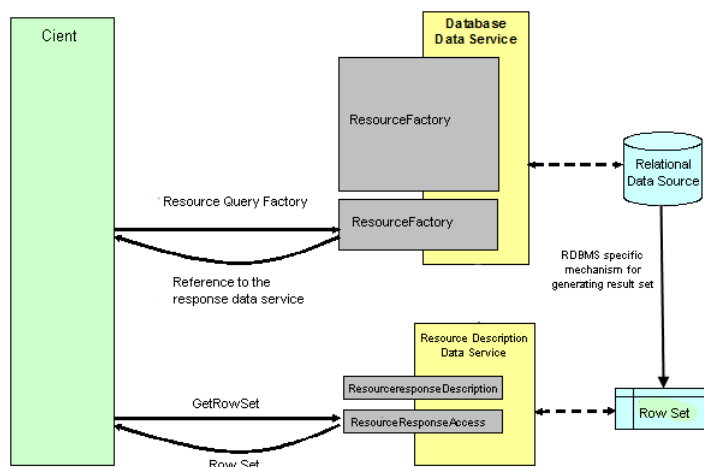


Figure 5.1: Direct and indirect access modes in WS-DAI

exist outside the service-oriented middleware and its lifetime is managed in ways that are specified in the DAIS specifications).

There are two different ways for accessing data resources. The first one is direct access to data resources with which the user accesses these resources like a regular service. The user sends a request to the resource with a SQL query in the parameters. The result of the web service is a row set with data requested.

The second way for accessing data resources is the indirect access. Indirect access implements a factory pattern for the user's data requests. Figure 5.1 represents the situation in which a user queries a data resource where the query results will be populated incrementally when they are available, following the previous indirect access. The WS-DAI specification does not define any operation for implementing the indirect access but defines the template to be followed. The details are then specified by the realisations that specialise the interface to the type of data being accessed.

Web service data access is a technology that can be employed for federating access to distributed data sources. It is widely used in workflow systems like [DBG⁺, HWS⁺, Ant07] in e-Science applications and for federating queries to databases [DT10].

5.2 Introduction to Distributed Query Processing

In this section we describe the general architecture of our system. We describe main architectural decisions and also a brief description of the systems that we extend, namely OGSA-DAI and its distributed query processor OGSA-DQP.

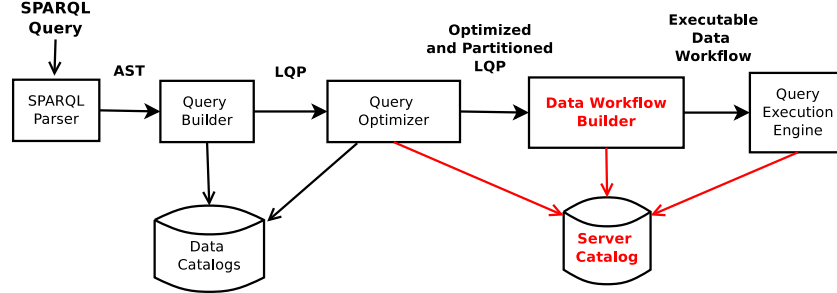


Figure 5.2: Generic Architecture for SPARQL-DQP

In Figure 5.2 we show a diagram containing the general architecture for SPARQL-DQP. This diagram is an adaptation of the diagram presented in [Kos00], also presented in Figure 2.6. In this figure there are five main components that process SPARQL 1.1 federated queries. First, a SPARQL query is parsed by the SPARQL parser, which transforms that query into an Abstract Syntax Tree (AST). This transformation allows us managing the query in a structured way, so that each element corresponds to a node in a tree. This tree is used by the Query Builder to build a Logical Query Plan (LQP) that represents the query using the SPARQL operators (OPTIONAL, Join, UNION, etc.). In the next step, the Query Optimiser optimises that LQP. Normally, for optimising a LQP data catalogues containing information about the data resources are used. In our case we do not use such catalogues since we assume that they are not provided to us. A difference from the original picture is that our optimiser is accessing the data server (nodes) information. Our system has a catalogue of available servers in which different parts of our query can be executed. These data nodes provide the information for distributing and parallelising the query processing. The output of this phase is an optimised logical query plan, which serves as input for the Data Workflow Builder component. This component builds the data workflow using the optimised query plan coming from the optimiser. It uses the information from the catalogue to configure the data workflow. It submits this to the query execution engine, which is the component that will actually perform and coordinate the execution.

Some of the components were present in the system before we extended it for distributing SPARQL queries. The system for executing data workflows is OGSA-DAI[Ant07] and the distributed query processor we extend is OGSA-DQP[ea09].

5.2.1 OGSA-DAI

OGSA-DAI¹ is an extensible framework that allows access (access to structured data in distributed heterogeneous data resources), transform (transform data in schema X to users as data in schema Y), inte-

¹<http://www.ogsadai.org.uk/>

5. ACCESSING DISTRIBUTED RDF DATA STORES

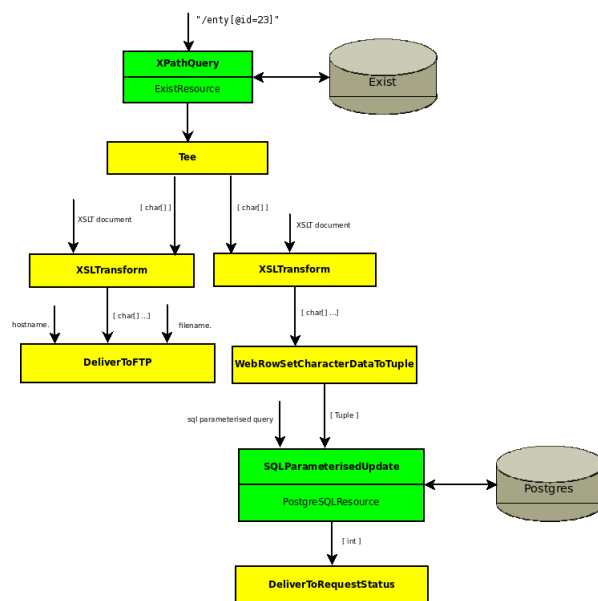


Figure 5.3: Simple example of an OGSA-DAI workflow

gration (expose multiple databases to users as a single virtual database, otherwise known as a federation) and delivery (deliver data to where it's needed by the most appropriate means, e.g. web service, e-mail, HTTP, FTP, GridFTP) of distributed data resources. The data resources supported by OGSA-DAI are relational databases, XML databases and file systems.

These features are collectively enabled through the use of data workflows which are executed within the OGSA-DAI framework. The components of the data workflows are activities: a well-defined functional units (data goes in, the data is operated on, data comes out) and can be viewed as equivalent to programming language methods. Data is streamed between activities so these data can be consumed by the next activity on the workflow as soon as it is outputted. The following Figure 5.3 represents a data workflow in which data resources are accessed, transformed and delivered.

In this example, a query to an XML database located at node X is sent. The "XPathQuery" activity is in charge of sending the query to the correct data resource representing the XML database. The output of this query is duplicated by the "Tee" activity. One of these outputs is delivered by the "DeliverToFTP" activity to a remote FTP server meanwhile the other output is processed converting it to the OGSA-DAI's tuple format by the "WebRowSetCharacterDataToTuple" activity. These tuples are then stored in a Postgres database located at node Y using a SQL update activity. The results are also delivered to a request status by the "DeliverToRequestStatus" activity to a user in node Z.

5.2.1.1 OGSA-DAI workflows

An OGSA-DAI workflow is a set of chained activities (or as said before functional units) which are executed in the main OGSA-DAI server. An OGSA-DAI Web service (data request execution service or DRES) receives the input workflow and this service provides access to a data request execution resource (DRER) which will parse the workflow, instantiate the activities in the workflow, execute it, build the status of the execution and return that status to the client. There are two different workflow execution modes:

Synchronous execution the data request execution service returns a request status to the client only when the workflow has completed execution. The data is as well returned if that is what is being asked for.

Asynchronous execution the data request execution service returns a request status to the client as soon as the workflow starts executing. Along with this request status, will be the ID of a request resource which the client can use to monitor the progress of their workflow.

One of the key features of workflow execution are that all activities in a workflow are executed in parallel, data streams go through activities in a pipeline-like way (as soon as a data unit is processed by an activity this data unit is buffered or sent to the next activity in the pipeline) and each activity operates on a different portion of a data stream at the same time. At this point pipeline buffers between OGSA-DAI activities are of key importance, since its configuration will allow better management of the data transfer and the parallelisation when communications are unstable. The default buffer size is of 20 OGSA-DAI tuples. This allows efficient processing of large data volumes as well as reduced memory footprints.

The other key element of the OGSA-DAI workflow processing system is the use of data streaming between activities. This workflow system is designed to stream data through activities in a pipelined fashion [Ant07]. If the activities are well-defined in the OGSA-DAI workflow, they can be executed concurrently. Each activity will process a different portion of the data stream, allowing to efficiently process large data with a small memory footprint. Lists and data grouping is also possible in OGSA-DAI allowing to better manage the transfer of binary large objects (BLOBs) and binary data.

5.2.1.2 OGSA-DAI Resources

OGSA-DAI Resources are named components which can be accessed, or referred to, by clients. When clients submit a workflow, that workflow is executed by the data request execution resource. This

5. ACCESSING DISTRIBUTED RDF DATA STORES

resource is the component that handles the workflow execution in OGSA-DAI. Clients will also make use of data resources. Data resources are an abstraction of databases or other data resources and will reference these in their workflows. However, there are more types of OGSA-DAI resources:

Data request execution resource (DRER) A data request execution resource executes OGSA-DAI workflows (also termed requests) provided by clients. A DRER can execute a number of such requests concurrently and can also queue a number more.

Data resource A data resource is an OGSA-DAI abstraction of a database or other type of data source.

Data sink A data sink is an OGSA-DAI resource which allows clients to push data to an OGSA-DAI server.

Data source A data source is an OGSA-DAI resource which allows clients to pull data from an OGSA-DAI server.

Session A session is an OGSA-DAI resource which acts as a state container associated with a sequence of workflows.

Request resource A request resource is an OGSA-DAI resource which allows the management of a request submitted to a DRER.

Resources are exposable by a WSRF-compliant [Cza04] presentation layer. Thus, OGSA-DAI resources have identities, properties and lifetimes that can be exposed by web services. An OGSA-DAI resource is an encapsulation of OGSA-DAI state and behaviour. A resource can be viewed as analogous to an object in object-oriented programming. Resources hold their state, can expose some of its state using resource properties, can be dynamically created or have associated lifetime management operations. OGSA-DAI resources also contains specific functionalities to the type of the resource (i.e. relational databases have specific resource properties different from a file system resource).

5.2.1.3 OGSA-DAI Web services

OGSA-DAI exposes resources via a web services presentation layers. These layers are based on the WSRF [Cza04] specification. WS-Addressing [MG06] provides a way for identifying OGSA-DAI's resources and operations to these resources, lifetime management and resource properties are managed by WS-ResourceLifetime [LS06] and WS-ResourceProperties [TB06] specifications, clients can access OGSA-DAI resources by using a resource ID or a WSRF compliant endpoint reference.

5.2.2 OGSA-DQP

The distributed query processor (DQP) is a set of resources and activities within the OGSA-DAI framework that execute SQL queries on a set of distributed relational databases managed by OGSA-DAI. Figure 5.4 shows how queries are processed and executed by the DQP coordinator.

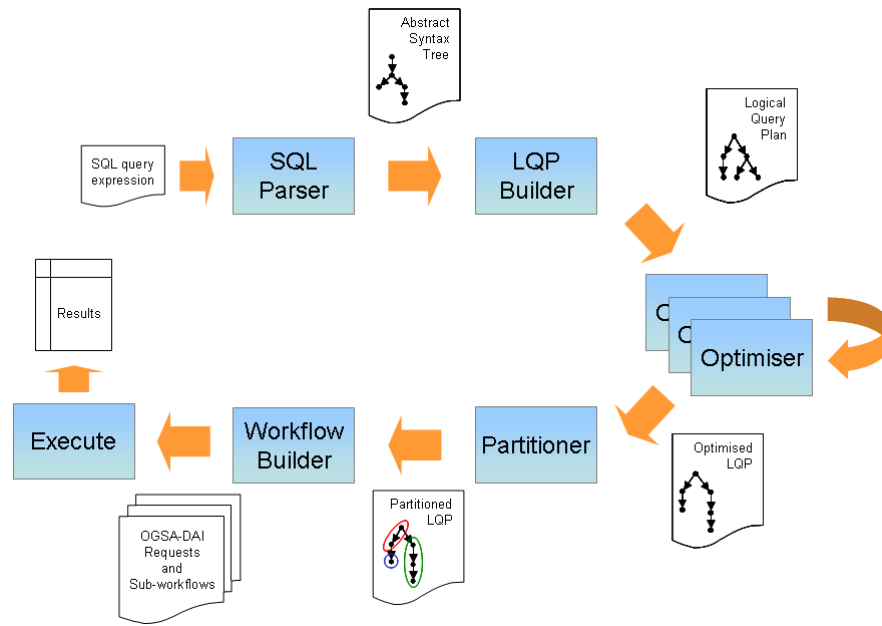


Figure 5.4: Processing of an SQL Query in DQP

5.2.2.1 DQP Resource

A DQP resource represents a coordinator of a distributed query processor and contains all the components enumerated below. It provides access to a set of distributed data nodes checking first the availability of the data nodes and extracting the table schemas that populate the DQP *data dictionary*. The data dictionary contains a view of the tables on the all the data nodes as a single database.

5.2.2.2 Data Dictionary

The data dictionary provides access to table schemas and other configuration data. The contents are static in the data dictionary implementation and initialised at startup time. In the future it may support storing and provide access to logical and physical metadata for the data nodes and other information that may be of use to query optimisers.

5. ACCESSING DISTRIBUTED RDF DATA STORES

5.2.2.3 SQL Expression Parser

The SQL expression parser supports a subset of SQL 92 as well as user defined functions (UDF) which were introduced as an extension point for the DQP query language. The ANTLR parser generator¹ is used for generating an SQL parser from a grammar. The parser produces an abstract syntax tree (an internal representation of the SQL expression) which is then passed on to the *LQP Builder*.

5.2.2.4 Logical Query Plan (LQP) Builder

The LQP Builder takes the abstract syntax tree generated by the SQL parser and produces a logical query plan. A logical query plan is a directed graph whose nodes are relational operators. A pluggable chain of optimisers may then be applied to this LQP, for example heuristic algebra-based optimisations or cost-based optimisations. In this context, the availability physical and logical database metadata and table statistics is crucial for query optimisation.

5.2.2.5 Query plan partitioning

If a query plan contains a join or product of two data streams that are located on different data nodes the partitioning algorithm detects this and transforms the LQP by inserting exchange operators that represent data transfers between two remote data nodes. The output from the partitioner is a set of partitions and the LQP in which every operator is annotated with the partition to which it belongs.

5.2.2.6 Query plan to OGSA-DAI workflow translator

The translator converts the partitioned LQP into a set of OGSA-DAI workflows. Each partition corresponds to a workflow and workflows may be connected by data exchanges. A workflow can be executed remotely on another OGSA-DAI data request execution service, or locally as a sub workflow within the same OGSA-DAI service. The workflows always result in a single data stream which is delivered to the coordinator.

An exchange of data between two workflows is realised as a combination of writing to an OGSA-DAI DataSource and transferring the data using an ObtainFromDataSource activity. The producer workflow writes its data to a DataSource and the consumer workflow transfers the data from the DataSource to its local node.

¹<http://wwwantlr.org/>

5.2.2.7 Workflow execution

In the final stage of the query processing the generated remote requests and local sub workflows are executed and the results collected and returned by the activity.

5.2.2.8 OGSA-DAI activities

OGSA-DQP provides a set of activities for performing query federation. These activities are specific to OGSA-DQP but they can also be used independently of the federation system. This allows extra extensibility for the system.

SQLQuery To a client, this activity behaves like an SQL query activity on a relational resource. The client provides an SQL query expression as input and a list of result tuples is returned. In the implementation of this activity, the SQL expression is passed to the DQP and processed as shown in figure 5.4 and described above. The results are collected by the activity and written to the output.

ExtractTableSchema This activity targets the DQP resource and extracts table metadata from the data dictionary, showing a view of the available tables as a single database. To avoid name clashes, tables are prefixed with the names of the data resource to which they belong.

TupleArithmeticProject This activity projects values from the input columns according to specified arithmetic expressions to output columns.

TupleSelect This activity selects tuples from a data stream which satisfy a Boolean expression. This activity usually corresponds to the WHERE or HAVING clauses of an SQL expression, or the relational SELECT operator.

TupleProduct This activity produces the direct product of two tuple input streams.

TupleLeftOuterJoin This activity performs a left outer join of two input data streams.

TupleUnionAll This activity performs a union of two input streams, without removing duplicates.

MetadataRename This activity modifies the metadata of an input data stream. This corresponds to renaming a column or a table using AS in an SQL expression.

Sort This activity is an implementation of a simple sort algorithm which also supports sorting of large data sets that do not fit into memory by caching them in a file.

5. ACCESSING DISTRIBUTED RDF DATA STORES

GroupBy This activity groups input tuples by a set of grouping columns and evaluates SQL aggregate functions per group, such as COUNT, SUM, MIN, MAX or AVG. The implementation supports group-by on large data sets.

OneTupleOnly This activity fails if the input data stream contains more than one tuple and is used for verifying scalar subqueries in an SQL expression.

5.2.2.9 OGSA-DQP optimisers

OGSA-DQP provides a set of optimisers that optimise the OGSA-DAI workflows that represent the SQL query:

VisualiseOptimiser This optimiser generates a visualisation of a LQP in a specific time. Formally it does not optimise the query plan but it helps to visualise a query plan in a specific state.

DecorrelationOptimiser This optimiser pulls the predicate of a select operator up to the operator above it. The select operator and any project operators in between are then removed. The optimizer makes three changes, the select operator and any project operators are removed, and the predicate of the select operator is appended to the predicate of the upper semi-join operator.

SelectPushDownOptimiser This optimizer pushes as down as possible SELECT operators.

RenamePullUpOptimiser This optimizer pulls up RENAME operators. Rename operators rename variable name within a query plan for allowing variable name disambiguation.

ProjectPullUpOptimiser This optimizer tries to pull up project operations as much as possible in the query plan. Projects are used for reducing the amount of data transferred by removing variables but these variables can also be used by other operators. By pulling up Projects, we assure that variables will be cropped when necessary.

JoinOrderingOptimiser This optimizer orders joins in the query plan depending on the estimated cardinality of each join. Based on the statistics gathered for each operator a new optimal join order is calculated.

JoinAnnotation This optimizer annotates joins with information about which side of the join is the most appropriate to read first. Part of the join data is stored in memory and selecting the most appropriate side of the join to be read first helps in the memory management.

InsertProjectAfterGroupByOptimiser This optimizer inserts PROJECT operators after GROUP BY operators so that the PROJECTS can be pushed down as far as they can go.

ProjectPushDownOptimiser Optimiser to push projects down so the redundant rows are not extracted from the databases, transferred over the wire etc.

RemoveRedundantProjectOptimiser Optimiser to remove redundant PROJECT operators from the logical query plan. A PROJECT is redundant if it contains no arithmetic expressions nor functions (thus it contains only column names) and the PROJECT operator's heading has the same attribute list as its child operator's heading.

PartitioningOptimiser Partitioning optimiser annotates each operator with an evaluation node annotation on which it should be executed. Partition boundaries are marked by EXCHANGE operators which are in charge of redirecting data to the execution nodes. Exchange operators are annotated with the same evaluation node annotation as their parent (destination node). The execution engine expects a properly partitioned query plan. In such a query plan all operators have evaluation node annotations which are consistent within a partition. Partition boundaries are also defined by EXCHANGE operators. The EXCHANGE operator's child and parent have different evaluation node annotations. The EXCHANGE operator is annotated with the same annotation as its parent operator (destination node). All partitions can be remote - execution engine will add local partition that deals with final results transfers automatically.

The optimisers execution order plays an important role in the query optimisation system. In the optimisation operators list there are contradictory operators (like ProjectPushDownOptimiser and ProjectPullUpOptimiser). Optimisations need an order to be applied so first, a normalised form of the query plan is generated for later on operating over this normalised query plan.

5.2.3 OGSA-DAI Limitations

The main limitations of OGSA-DAI are based on the mediator/wrapper layer. Why another layer is needed between clients and data? OGSA-DAI provides an abstraction layer of the data, which is accessed by OGSA-DAI servers. OGSA-DAI provides a Web service presentation layer (which is optional) for allowing clients to execute data workflows. This has performance implications, since invoking Web services for composing workflows has time implications (it adds the cost of invoking each Web service). Also, speed for accessing data sources may be paramount, since when they are accessed they have still to be transferred between the OGSA-DAI nodes.

5. ACCESSING DISTRIBUTED RDF DATA STORES

These are the main problems that we identified within OGSA-DAI. These drawbacks are compensated with robustness for executing data workflows. This robustness has already been proved in many projects like ADMIRE¹, GeoTOD-II², the BIRN - Biomedical Informatics Research Network³ and many others. Besides, the cost of invoking a set of Web services may be negligible when accessing large amounts of data.

5.3 Accessing Distributed RDF data

In this section we describe how to access RDF data. We distinguish between two different kind of RDF data sources. These two sources are relational databases that provide RDF access via ontology mappings and data directly formatted in RDF. The former approach for accessing data in the RDF data model focuses in providing an ontological layer to a non ontological data resource. The data resources can be of different types (i.e. lexicons, thesauri, XML files, relational databases, etc.). We will focus on accessing relational data resources, i.e. RDB2RDF systems. The latter approach is data directly formatted in RDF. These data is stored in RDF databases that offer an entry point via the SPARQL protocol called SPARQL endpoint.

5.3.1 Accessing RDF data sources

We build an extension to OGSA-DAI which consists in adding a new type of data resource that accesses RDF datasets. This RDF data resource provides access to an RDF stores that offer its data by means of SPARQL endpoints. This resource is configured with the URL of a SPARQL endpoint to which the query is addressed, together with other lifetime properties.

Figure 5.5 shows a conceptual view of the new data resource. In this figure, a SPARQL endpoint has been added to the other data sources accessed by OGSA-DAI (relational databases, XML databases and indexed files). Clients access these data sources by creating an OGSA-DAI workflow, which will access and process the data (if indicated so).

Queries are sent to this new type of RDF data resource by means of OGSA-DAI activities. The implementation of the RDF data resource sends the query to the corresponding SPARQL endpoint and then waits for the results. These results can be directly returned to the requester or kept at the server wrapped up as a new data resource, following the direct and indirect access modes respectively. These

¹<http://www.admire-project.eu/>

²<http://tiger.dl.ac.uk:8080/geotodls/index.htm>

³<http://www.birncommunity.org/>

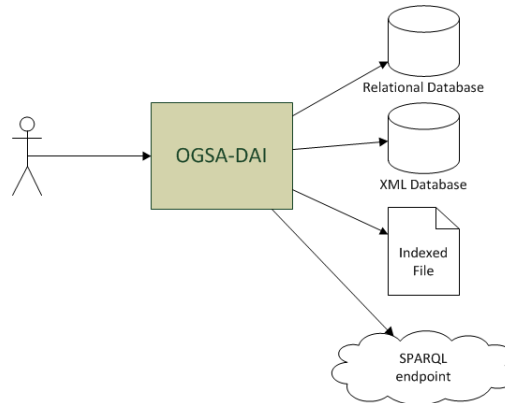


Figure 5.5: OGSA-DAI RDF extension

results are provided as RDF data streams, using the internal data representation used by OGSA-DAI, which allows faster communication between nodes in distributed settings.

The data resource architecture contains a set of access providers at server side and a set of activities that allow users to compose data workflows:

RDFResource This class implements the OGSA-DAI `DataResource` interface. The resource class manages the resource configuration. It provides access to the configuration parameters, in this case, the configuration parameters of an RDF database. This class is invoked at the system's initialisation so it will be available when an OGSA-DAI activity requests data. The implemented methods are `getState()`, which returns the configuration parameters of the "RDFResource", including the necessary elements for the correct execution of the data resource; `initialize()`, which creates the RDF data resource and initialises it allowing the OGSA-DAI persistence and configuration components to configure the data resource class with its configuration (the SPARQL endpoint address). Other methods for managing the life cycle of the resource are also implemented in this class.

RDFConnectionProvider This class manages the connection to the RDF resource. When requested, this class provides the necessary elements for accessing the data resource. In this case, these elements are the database connection parameters necessary to access it.

RDFResourceState This class represents the state of the RDF data resource. On data resource creation the associated `DataResourceState` (in this case the RDF resource) object is registered with OGSA-DAI persistence and configuration components. Any changes to this object are persisted. Hence if the configuration of a data resource is to be persisted it should be reflected in the associated `DataResourceState`.

5. ACCESSING DISTRIBUTED RDF DATA STORES

5.3.1.1 RDF Activities

New OGSA-DAI activities have been created to allow access of RDF data resources. We created two of them for accessing RDF data, one for accessing the RDF data resource (either in an RDF database or in a remote SPARQL endpoint) and another for accessing a SPARQL endpoint directly. With the latter activity we provide a mean for clients to specify any SPARQL endpoint in the query without the need of having an OGSA-DAI resource representing the data source.

RDF server activity This activity is in charge of connecting the client side activity and the RDF data resource. Not only it does it connect the client to the data resource but it also creates activities that manipulate the outputs from the resource if necessary. The activity connects to the resources, obtains the configuration parameters and queries the remote RDF data set. The activity extends the ActivityBase class, which is responsible for providing access to input and output pipes, activity contracts and simple validation functionality. For interacting with the RDF resource the activity has to implement the ResourceActivity class and has to contact the OGSA-DAI resource accessor.

Direct RDF server activity This activity does not need an RDF resource to query a remote RDF data source. Instead, it connects to the OGSA-DAI's DataRequestExecutionResource, which is a resource containing the default data manipulation activities. This activity has the advantage of querying directly the remote SPARQL endpoint without having to configure a RDF resource. Its drawback is that it is not possible to access RDF databases since there is no configuration file available.

RDF client activity The client side activity connects to the server side activity, requesting the execution of an RDF query. The activity gets as response a pointer to where the results are being stored. Once the results are available it is possible to compose the output of the activity with other outputs, create workflows, etc.

The processing of the RDF result set, converting the SPARQL results into OGSA-DAI tuples, is done by the RDFUtilities class. This class is contacted by the RDF server side activity once the query has been executed. Then, the results of that query are streamed into the conversion process and later on streamed out to the next workflow activity.

5.3.2 Accessing RDB2RDF data resources

Figure 5.6 show the next extension to OGSA-DAI that we developed. In this case, the data source accessed is a RDB2RDF system. In the same way than we extended OGSA-DAI for accessing RDF data

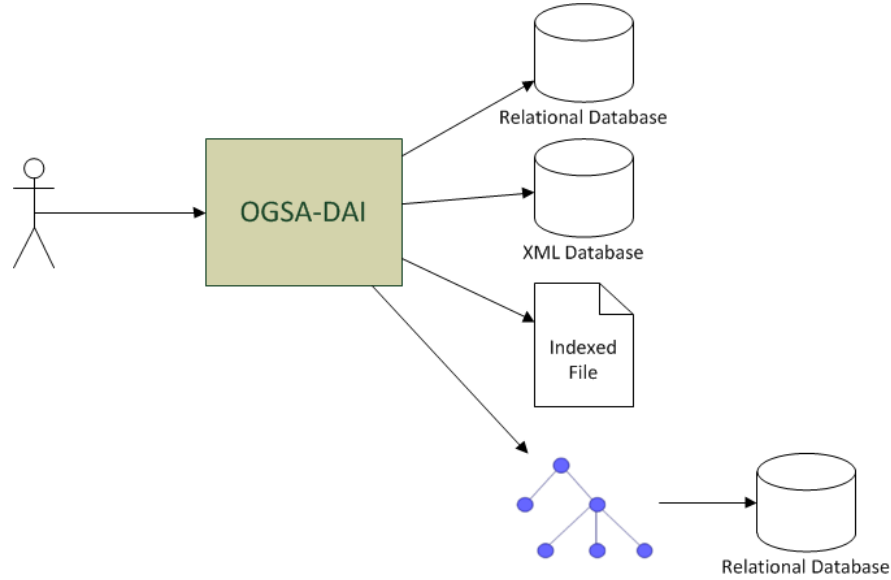


Figure 5.6: OGSA-DAI's RDB2RDF extension

(described in Section 5.3.1: clients can now access RDB2RDF resources by creating a data workflow which will be send to OGSA-DAI.

RDB2RDF systems, as explained in Chapter 2 are systems that provide a processor (mediator system) between a relational database and a user. The mediator is in charge of translating the SPARQL query to the relational database language. For accessing such systems, we extended OGSA-DAI [Ant07] with a new type of resource which represents the RDB2RDF processor. This resource in our case is the ODEMapster engine, which processes R2O mapping files [BCGP04]. The ODEMapster resource is configured for accessing the relational database through a mapped ontology using the R2O language. We refer the reader to Section 2 in where a detailed description of R2O and ODEMapster is provided.

The design of the new RDB2RDF data resources follows the architecture of other data resources in OGSA-DAI. The resource represents the data source, which contains all parameters for accessing it, once deployed in the system

5.3.3 R2O resource

As we said before, the new data resource (from now on R2O resource) follows the other OGSA-DAI's resources design principles. The data resource itself is a configuration file with all the needed configuration parameters for accessing the data. These configuration parameters may include security parameters, data location, connection types, etc. The class that accesses this file configuring the system implements the "uk.org.ogsadai.resource.dataresource.DataResource" class. Thus, when the whole OGSA-DAI sys-

5. ACCESSING DISTRIBUTED RDF DATA STORES

tem is initialised, the resource class (in our case the R2OResource class) accesses the configuration file and creates its state in the system. The state contains all the parameters in the resource configuration file plus others like lifetime properties as defined by [Cza04]. The R2O resource will be accessed when necessary by a connection manager (called ConnectionProvider) and the data resource state (ResourceState):

R2OConnectionProvider This class is in charge of managing the connection between the clients willing to access the data resource and the resource itself. It provides methods for accessing the data (a relational database in this case), releasing the data resource, etc.

R2OResourceState This class current the state of the R2O data resource for a specific client. On data resource creation the associated DataResourceState (in this case the R2O resource) object is registered with OGSA-DAI persistence and configuration components. Any changes to this object are persisted. Hence if the configuration of a data resource is to be persisted it should be reflected in the associated DataResourceState. The data resource contains specific parameters for that specific execution.

The ODEMapster processor, in charge of executing the transformation of the SPARQL queries into SQL queries is implemented by the MapsterConnector class. It processes the R2O mapping file by running the ODEMapster engine. This class accesses the R2O resource state and its connection provider for the processing of the mapping files. However, the invocation of the Mapster processor is done by OGSA-DAI's activities.

5.3.4 R2O Activities

OGSA-DAI activities are the OGSA-DAI key elements. Combining activities is possible to build data processing workflows. OGSA-DAI's activities access the OGSA-DAI resources, in our case, by activating the ODEMapster processor. There are two kinds of OGSA-DAI activities, server side and client activities:

R2O server activity this activity is in charge of connecting the client side activity and the R2O data resource. Not only does it connect the client to the data resource but also it creates activities that manipulate the outputs from the resource. In our case the activity created only starts the engine that creates the instances in the ontology from the database. The activity extends the ActivityBase class, which is responsible for providing access to input and output pipes, activity contracts and

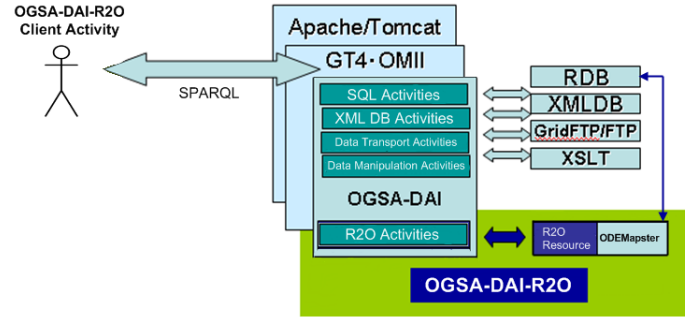


Figure 5.7: OGSA-DAI-R2O extension (based on OGSA-DAI-RDF extension figures)

simple validation functionality. For interacting with an R2O resource the activity has to implement the ResourceActivity class and has to contact the OGSA-DAI resource accessor.

R2O client activity The client side activity connects to the server side activity, requesting the execution of the R2O processor. This activity is part of a programmatic representation of the data workflow that is to be executed at the server side. The activity gets as response a pointer where the results are being stored. Once the results are available it is possible to compose the output of the activity with other outputs, create workflows, etc. Figure 5.7 shows the interactions of the activities (client and server side activities) with an R2O resource and its location within OGSA-DAI and the possible servers.

5.3.4.1 Execution

The user starts their application by invoking OGSA-DAI by sending a DataRequestExecutionResource. This is the resource in charge of calling resources that need to be executed. In our case the "R2OResource" will be invoked. Now the server receives the request from the client and invokes the R2OActivity at the server side. This activity communicates with the R2OResource invoking the getR2OResult() method making ODEMapster run the R2O mappings to generate the RDF instances from the relational database. This activity also manages connections to the databases and could be extended to have more functionality (which is detailed in the future work section).

5.4 SPARQL-DQP: Enabling Distributed SPARQL Query Processing in OGSA-DQP

Now that we have discussed how we have enabled several types of RDF resources in OGSA-DAI, we move into the implementation of our distributed query processor for SPARQL.

5. ACCESSING DISTRIBUTED RDF DATA STORES

From a high level point of view, SPARQL-DQP can be defined as an extension of OGSA-DQP that considers an additional query language: SPARQL. The design of SPARQL-DQP follows the idea of adding a new type of resource to the standard data resources provided by OGSA-DAI (relational databases, XML databases and file systems), and extending the parsers, planners, operators and optimisers that are used by OGSA-DQP to handle this new type of resource.

Therefore our first extension to OGSA-DAI consists in adding a new type of data resource that accesses RDF datasets. These RDF extensions were described in sections 5.3.2 and 5.3.1. The RDF data resource provides access to RDF or RDB2RDB data stores that offer their data by means of SPARQL endpoints. Queries are sent to the new RDF data resource by means of OGSA-DAI activities. The implementation of these data resources sends a query to the corresponding RDF data set and waits for the results. These results can be directly returned to the requester or kept at the server wrapped as a new data resource, following direct and indirect access modes respectively. The results are provided as RDF data streams, using the internal data representation used by OGSA-DAI, what allows faster communication to take place between nodes in a distributed setting.

Next we extend OGSA-DQP to accept, optimise and distribute SPARQL queries across different data nodes. SPARQL-DQP adds new parsers, LQP builders, operators and optimisers so as to read the query, create a basic query plan, optimise it, partition it and send it to the different nodes in which the different parts of the query will be processed.

Figure 5.8 shows the SPARQL-DQP architecture within OGSA-DAI. The following sections discuss its main components.

5.4.1 SPARQL-DQP Resource

A SPARQL-DQP resource represents the coordinator of the distributed query processor, as in OGSA-DQP. This coordinator extends the original OGSA-DQP coordinator in such a way that the data resource is essentially the same but changing the classes that perform the federation and the data dictionary. At initialisation time the SPARQL-DQP resource checks the availability of the data nodes in which the federation will be executed (not the SPARQL endpoints provided to access the RDF data) and extracts remote RDF resources information. The extraction is done by the SPARQL federation system, which obtains the necessary information for the SPARQL federation. This information is stored in the SPARQL-DQP *data dictionary*. The data dictionary contains a description of all available characteristics of the various data nodes. These characteristics are information about ad-hoc functions implemented by the remote RDF resource, data node information (like security information, connection information

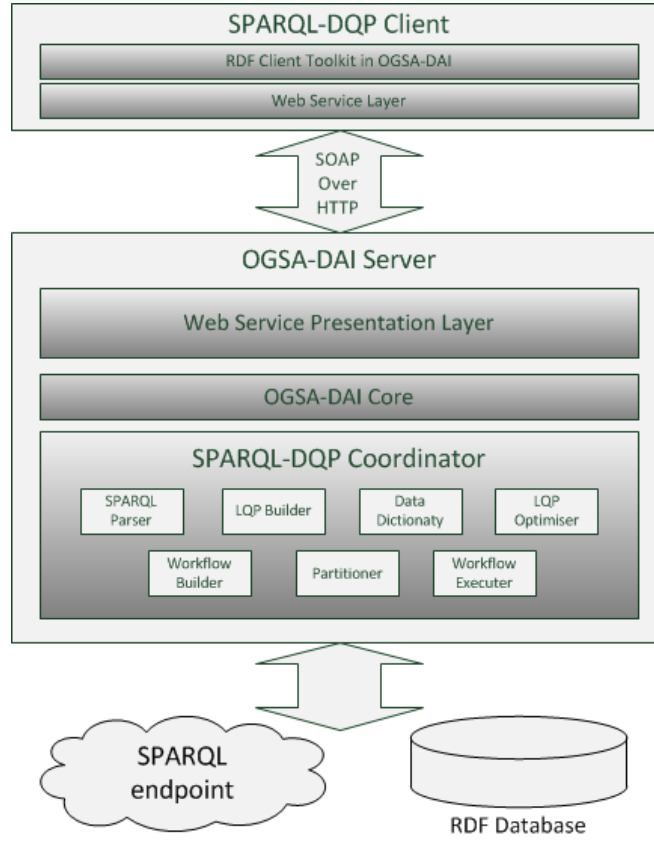


Figure 5.8: SPARQL-DQP Architecture

and data node address) and table metadata (currently only the RDF repository name, to be extended with statistics about the data in the datasets).

5.4.2 RDF Data Dictionary

The data dictionary interface, as in OGSA-DQP, provides access to table schemas and other configuration data. The main differences between the original SQL data dictionary our data dictionary are:

- The RDF data dictionary stores information about the execution nodes.
- The RDF data dictionary only contains information about the remote SPARQL endpoint or RDF database, since the data model is always the same: ?s, ?p, ?o

The data dictionary is used to provide the necessary information to build the SPARQL *logical query plan* (LQP). The logical query plan built by the SPARQL LQP builder, which composes the initial query plan that will later on be transformed into a data workflow. The contents of the RDF data dictionary are

5. ACCESSING DISTRIBUTED RDF DATA STORES

mostly static in this implementation and initialised at startup time. The static contents are those contents that are provided by the RDF data resources. As we described in Section 5.3.1, RDF data resources access RDF databases and configuration parameters that correspond this data resource. Dynamic dictionary content loading takes part when direct queries to remote SPARQL endpoints are issued. These queries have no a priori knowledge of the remote SPARQL endpoint, and a dynamic loading of the server data that is going to execute this query is needed.

5.4.3 SPARQL Expression Parser

The SPARQL expression parser supports a subset of SPARQL 1.0 plus the SPARQL 1.1 Federation Extension. Also, specific user defined functions (UDF) can be introduced as an extension point for the SPARQL-DQP query language. SPARQL 1.0 implementations (like Virtuoso) implement extensions to the language to facilitate access to RDF data. Functions like `bif:contains` which is used as an indexed term search can be supported by UDFs. Formally, UDFs allow developers to add support for arbitrary functionality within the SPARQL-DQP.

The SPARQL expression parser also uses the ANTLR parser generator¹ which generates a SPARQL parser from the SPARQL 1.0 grammar plus the Federation extension. The parser produces an abstract syntax tree (an internal representation of the SPARQL expression) which is then passed to the *SPARQL-LQPBuilder*.

5.4.4 SPARQL Logical Query Plan Builder

The SPARQL LQP Builder takes the abstract syntax tree generated by the SPARQL parser and produces a logical query plan. The logical query plan follows the semantics defined by the SPARQL-WG² in the SPARQL 1.0 specification [PS08] which are also formalised in Chapter 2. The query plan produced represents the SPARQL query using a mix of operators and activities coming from the existing ones in OGSA-DQP and newly added for the specific characteristics of SPARQL. These new operators and activities are described below.

The same chain of optimisers that was described for OGSA-DQP is applied here, but we added rewriting rules based on well-designed pattern based optimisations. Besides, safeness rules have to be checked as we described in 4.4.1, since some SQL optimisers can only be applied to safe SPARQL patterns. The new optimisers are also described below.

¹<http://www.antlr.org/>

²<http://www.w3.org/2009/sparql/wiki/>

5.4.5 Query plan partitioning

The same algorithm that is used to partition the logical query plan for OGSA-DQP is used here but with we can now dynamically access remote nodes. In the original partitioner, the data dictionary contains a static list of data nodes in which the execution of the federation happens. This version of the partitioner accesses a dynamic list of execution nodes that vary across different query executions. As we explained in 5.3.1 SPARQL endpoints do not have a data node associated with themselves, instead they associate to available evaluation nodes and are attached to the data request execution resource. Thus the federation may vary according either to the data nodes from which the datasets originate or the available evaluation nodes.

5.4.6 Query plan to OGSA-DAI workflow translator

The translator converts the partitioned LQP into a set of OGSA-DAI workflows. Each partition corresponds to a workflow and workflows may be connected by data exchanges. A workflow can be executed remotely on another OGSA-DAI data request execution service, or locally as a sub workflow within the same OGSA-DAI service. The workflows always result in a single data stream which is delivered to the coordinator.

5.4.7 Workflow execution

In the final stage of the query processing the generated remote requests and local sub workflows are executed and the results collected and returned by the activity.

5.4.8 New OGSA-DAI activities

The following activities and operators were implemented for SPARQL-DQP:

SPARQLQuery To a client, this activity behaves like an SQL query activity on a relational resource.

The client provides an SPARQL query expression as input and a list of result tuples is returned. In the implementation of this activity, the SPARQL expression is passed to the SPARQL-DQP and processed as shown in figure 5.4 and described above. The results are collected by the activity and written to the output.

OptionalJoin This activity implements a SPARQL OPTIONAL operator. The semantics of this operator are the same than the previous OptionalJoin performing a left outer join of two input data streams, but with a different implementation. It extends a ThetaJoin and removes specific unused methods.

5. ACCESSING DISTRIBUTED RDF DATA STORES

TupleSparqlUnion This activity performs a SPARQL union. As described in Section 2, the SPARQL union differs from the SQL union and a new activity had to be created. The SPARQL union is the set theoretic union while for the SQL union operator to work both schemas (tables) have to be compatible. The implementation of this new activity stores first results from one schema in an index to quickly access them when unioning with the other schema values later.

RDFServiceScanOperator This operator is equivalent to a SQL scan operator. It performs a scan of the remote RDF database or SPARQL endpoint, using the SPARQL query in the SERVICE as the scanning query. The address is obtained from the SERVICE IRI and the results to that scan contain all variables in the scan query bounded.

ServiceVarOperator This operator executes a SPARQL SERVICE VAR pattern, performing an RDF scan over all the IRIs pointed to by the results of the variable in the pattern. The way this work is similar to the previous RDFServiceScanOperator but iterating over the results of the service variable and performing the union of the results, as was defined in Section 4.3. We also assure that the query can be safely executed by providing an execution order.

We also implemented a set of user defined functions (UDFs) for covering The SPARQL set of operators and functions (regex, isIRI) used to construct constraints within the FILTER operator. The user defined functions are attached to a SELECT operator which acts as the filter executor.

FunctionRegex This function matches a given regular expression in the streamed results from the query. The regular expression syntax used by these functions is defined in terms of the regular expression syntax specified in XML Schema¹.

FunctionisBlank This function identifies if one result in the stream is a blank node.

FunctionIsURI This function identifies if a result is a URI.

FunctionIsBound This function identifies whether a result is bound or not.

FunctionLiteral This function identifies if a result is a literal.

¹<http://www.w3.org/TR/xpath-functions/>

5.4.9 SPARQL-DQP optimisers

In SPARQL-DQP there are two different sets of optimisers. The first set of optimisers contains those optimisers based on SQL. These optimisers were already developed in the OGSA-DQP and were described in Section 5.2.2.9.

We have implemented the rewriting rules described in Section 4.4.1 with a bottom up algorithm for checking the condition of being well-designed. Once a well-designed pattern has been identified (checked for safety and well-designed conditions) transformation rules are applied. These are implemented and made available as new optimisers in the overall SPARQL-DQP framework.

Well-designed pattern rule 1 This rule identifies the SPARQL pattern $((P1 \text{ OPT } P2) \text{ FILTER } R)$.

When the well-designed pattern is identified the rewriting rule is applied for transforming the pattern into $((P1 \text{ FILTER } R) \text{ OPT } P2)$. Notice that this transformation can only be done if the pattern is safe as defined in Section 4.4.1. This optimizer is equivalent to the `SelectPushDownOptimiser` described before since both optimisers place filtering operators as early as possible. `SELECT` operators are used for filtering results in SPARQL queries. Pushing down these operators as much as possible reduce the amount of data transferred between nodes. Pattern safeness (described in Section 4.2) is required in SPARQL queries in order to correctly apply this operator.

Well-designed pattern rule 2 This rule identifies and applies rule 2 of the well designed patterns described in Section 4.4.1. The SPARQL pattern identified is $(P1 \text{ AND } (P2 \text{ OPT } P3))$ and it is transformed in $((P1 \text{ AND } P2) \text{ OPT } P3)$. The safeness condition is also needed for applying the pattern reordering.

Well-designed pattern rule 3 This optimisation identifies the SPARQL pattern $((P1 \text{ OPT } P2) \text{ AND } P3)$ and transforms it into the pattern $((P1 \text{ AND } P2) \text{ OPT } P3)$. This rule is equivalent to rule 2 using commutativity of operands. The resulting pattern is equivalent to rule number 2.

Optimisations are applied in a sequential order. First, pattern reordering optimisations (reordering rules 1,2 and 3) are applied. We first apply these rules to provide the other optimisers a logical query plan which is optimised from a logical perspective, before applying any other physical optimization (like partitioning the query plan in the different nodes available). Rule 1 is applied first, in order to push down as soon as possible `FILTER` predicates, since they are the most restrictive operators. Next, `OPTIONAL` patterns are identified, and optimised using the reordering rules. In all three optimisers, safety conditions are checked first. Next, the other OGSA-DQP optimisers are applied.

5.4.10 Federated SPARQL query execution

Figure 5.9 shows a setup for a distributed query processing scenario. In this scenario a SPARQL query is sent to two different SPARQL endpoints and to two RDF databases. These SPARQL endpoints are represented by one single OGSA-DAI evaluation node (the evaluation node has a data request execution resource associated which is in charge of executing the SPARQL query), and the RDF databases are represented by two different data nodes, which have the corresponding RDF data resource deployed.

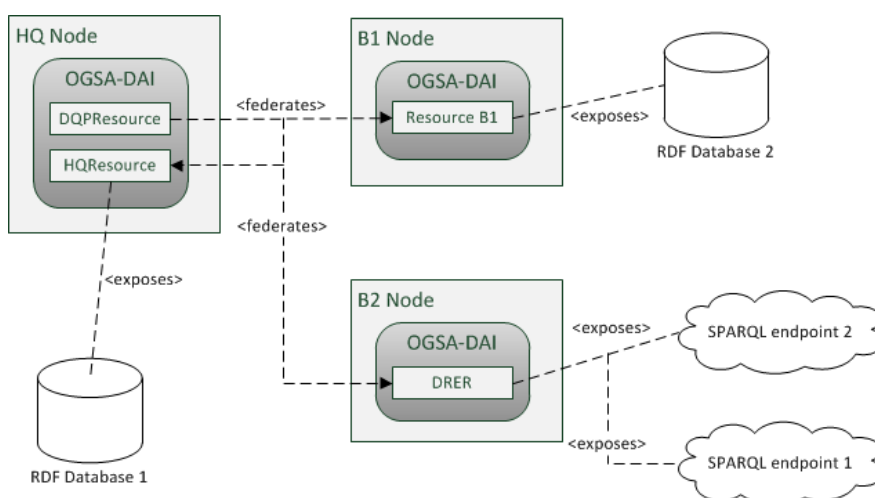


Figure 5.9: SPARQL 1.1 federated query execution

The sequence of operations is as follows: The SPARQL query is sent to the HQ node, which is the main OGSA-DQP server in this particular configuration. The other OGSA-DAI nodes (B1 and B2) can also receive SPARQL queries, the only difference is that the node receiving the query would act as main node in the federation. When the query is received, SPARQL-DQP parses the query and generates an abstract syntax tree. An AST sample is represented in Figure 5.10.

The AST serves as input for the SPARQL logical query plan builder. This builder transforms the AST into a logical query plan (LQP) that represents the SPARQL query. The algorithm for creating the AST is the algorithm described in the SPARQL 1.1 specification¹. A sample LQP is shown in Figure 5.11.

The next step is to optimise the LQP generated in the previous phase. In this optimisation phase all optimisers described for OGSA-DQP and the reordering rules implemented for SPARQL-DQP are applied. Also, the partitioning optimizer selects the best nodes to execute the data workflow represented by the optimised LQP.

¹<http://www.w3.org/TR/sparql11-query/>

5.4 SPARQL-DQP: Enabling Distributed SPARQL Query Processing in OGSA-DQP

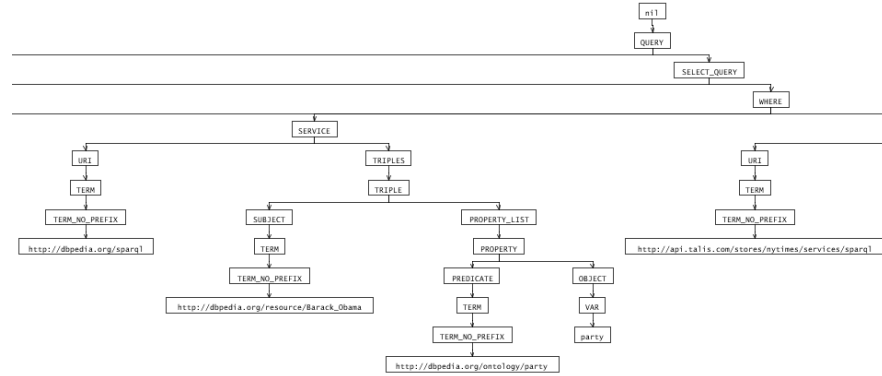


Figure 5.10: AST representing a SPARQL 1.1 federated query

The optimised LQP is sent to the OGSA-DAI workflow translator which converts the partitioned LQP in a set of OGSA-DAI workflows. These workflows have where they will be executed annotated in and how the streams of data have to be transferred between nodes and operators using the exchange operators.

Finally, the data workflow is executed, divided into generated remote requests. Sub workflows are executed locally and the results collected and returned by each node. The final results are sent to the client application. Notice that the client application has to build an initial data workflow for submitting a query. This data workflow is composed of a set of activities, including an activity that wraps the SPARQL query and a byte transformation activity to allow faster transmission of the data over the network. This activity also specifies the size of the SOAP messages in which data are to be wrapped. To choose the appropriate message size is important not to generate a high data overhead on the network.

5.4.11 SPARQL-DQP Limitations

SPARQL-DQP has the same limitations than OGSA-DAI, named that it contains another layer that may be useless in certain application domains and the use of Web services, that may increase the amount of time needed for returning results in certain query types. Regarding the first problem, our approach is clearly not needed for simple queries to SPARQL endpoints which limit their output data to a few numbers of results. A new abstraction layer implies a processing that can be done by more lightweight approaches. In this regard, the use of Web services is also not very useful for queries to these type of SPARQL endpoints, for the same reason: it is not necessary to manage the processing with complex data workflows, because they are designed for other purposes.

5. ACCESSING DISTRIBUTED RDF DATA STORES

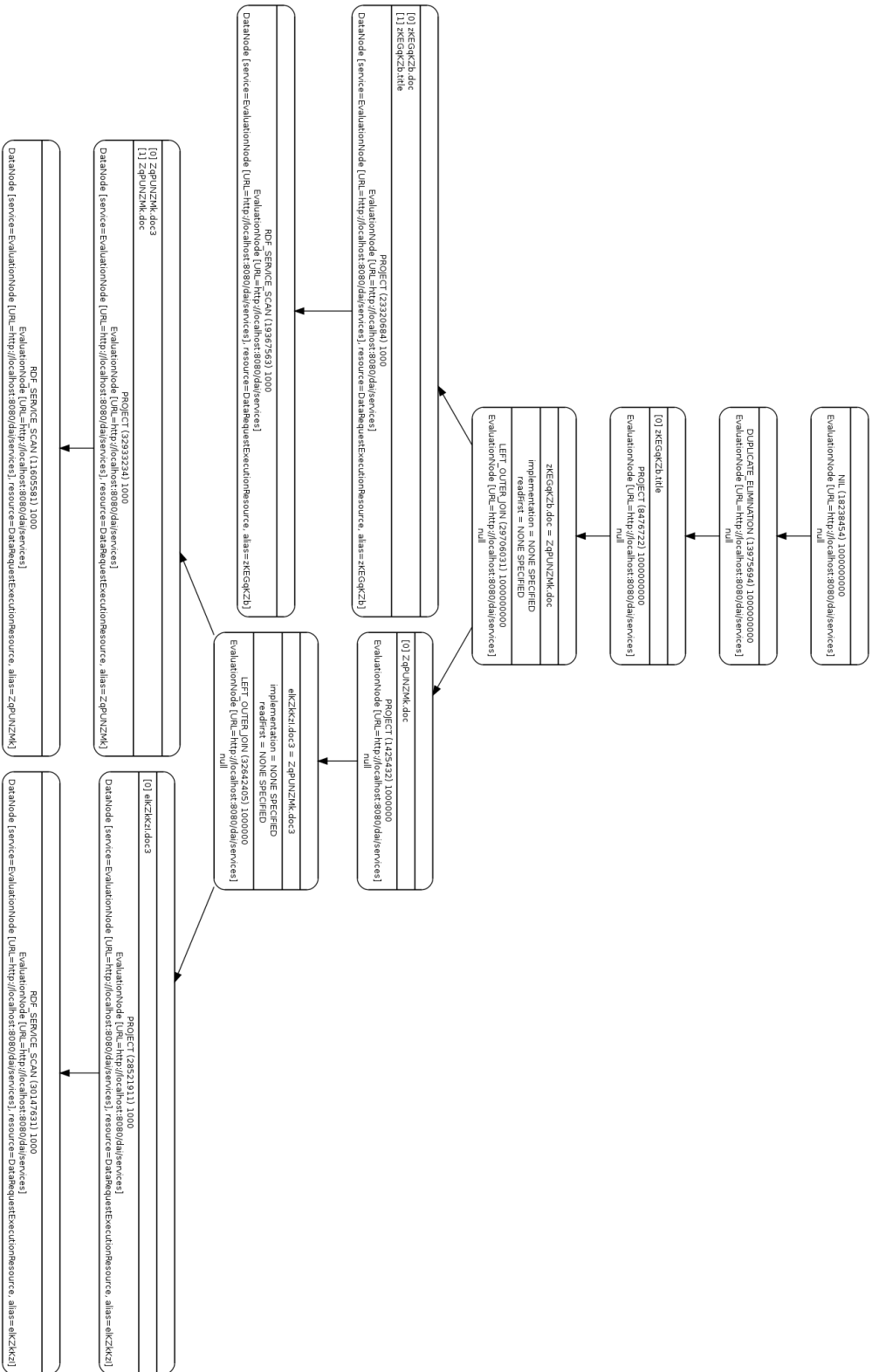


Figure 5.11: Logical Query Plan representing a SPARQL 1.1 federated query

5.4.12 SPARQL-DQP summarising

We saw in Section 2.4.1.1 a list of tables containing a summary of what technologies implemented the existing distributed SPARQL query processing systems. We performed a survey of the existing technologies for distributed query processing and optimisation and we related it to existing distributed SPARQL query processing systems. We now add our system, SPARQL-DQP to these comparison tables, so we can compare it with the existing systems in the state of the art.

Table 5.1: Data Integration Level in SPARQL-DQP

	Other Systems	...	SPARQL-DQP
Mediation	?	...	Yes
Federation	?	...	Yes
Mediation with update	?	...	No
Data Workflows	?	...	Yes

Table 5.1 shows the type of architecture for information/data integration that are implemented by the distributed SPARQL query processors. In our case we provide mainly a SPARQL federation extension for OGSA-DQP which is a distributed query processor that federates access to distributed relational databases. Since we base our system in OGSA-DAI/DQP we also provide a data workflow system that runs the actual federation. OGSA-DAI follows the mediator/wrapper architecture [Wie92b] since it allows integration of different data sources and uses wrappers for representing these heterogeneous data sources. Thus, our system also follows the mediator/wrapper architecture.

Table 5.2: Type of SPARQL-DQP's client-server Architectures

	Other Systems	...	SPARQL-DQP
Client-server	?	...	Yes
Middleware	?	...	Yes
P2P	?	...	No

Table 5.2 show that SPARQL-DQP implements. Since it extends OGSA-DAI/DQP SPARQL-DQP has the same architecture, which partially implements a strict client-server architecture from the user's point of view. Users request a query/workflow execution to the OGSA-DAI server and later on they receive the results of that workflow execution. From the server's point of view, there is a piece of middleware between the server and the final data sources that are integrated.

Table 5.3 presents the algorithms surveyed in Section 2.3.2.2. Of those, SPARQL-DQP implements a heuristic-based algorithm for generating an optimised query plan. It is based on the OGSA-DQP's

5. ACCESSING DISTRIBUTED RDF DATA STORES

Table 5.3: SPARQL-DQP's Query Planning Optimisations

	Other Systems	...	SPARQL-DQP
Heuristics-based	?	...	Yes
Dynamic Programming	?	...	No
Cost estimation	?	...	No
Response-time models	?	...	No

algorithm for query planning. It basically applies optimisers in a specific order until the final query plan is generated.

Table 5.4: SPARQL Specific Features in SPARQL-DQP

	Other Systems	...	SPARQL-DQP
SPARQL 1.1 Fed	?	...	Yes
Semantic Query Optimisation	?	...	No
Pattern reordering without statistics	?	...	Yes
Pattern Grouping	?	...	No

Table 5.4 shows that SPARQL-DQP implements the SPARQL 1.1 federation extension but it does not implement endpoint selection or pattern grouping, since both are already done by the W3C's recommendation. SPARQL-DQP reorders SPARQL queries without the need of statistics.

Table 5.5: Query Execution Optimisations in SPARQL-DQP

	Other Systems	...	SPARQL-DQP
Row blocking	?	...	Yes
Multicast optimisations	?	...	No
Multi-threaded	?	...	Yes
Horizontally Partitioned Data	?	...	No
Semi-joins	?	...	No
Double-Pipelined Hash Joins	?	...	Yes
Use of BINDINGS	?	...	No
Top-Bottom Optimisations	?	...	No
Streaming	?	...	Yes

Table 5.5 shows the query optimisation techniques implemented by SPARQL-DQP. These optimisations are mainly obtained by extending OGSA-DAI/DQP. SPARQL-DQP accesses the row blocking optimisation using OGSA-DAI. OGSA-DAI groups tuples into blocks of tuples of the same type. The tuple grouping into blocks is done by the activities used in the data workflow and can be configured by provid-

5.4 SPARQL-DQP: Enabling Distributed SPARQL Query Processing in OGSA-DQP

Table 5.6: Query or Data Transfer in SPARQL-DQP

	Other Systems	...	SPARQL-DQP
Query shipping	?	...	No
Data shipping	?	...	No
Hybrid-shipping	?	...	Yes

ing specific parameters within these activities. An example is the use of `TupleToWebRowSetCharArrays` for transferring activity results. OGSA-DAI also implements the use of multiple threads for processing data workflows and SPARQL-DQP takes advantage of this. OGSA-DAI uses a streaming model in which each activity is executed concurrently in a separate thread. When a query is evaluated by a single OGSA-DAI instance that federates over local resources, all of the activities will compete for CPU cycles [DT10]. OGSA-DAI provides a parallel hash join algorithm. This algorithm is based on the join activities available. These activities are two [Eke]:

Pipelined join that runs two threads, each processing one of the data streams. The threads buffer the tuples they have received and compare them to the tuples of other threads as tuples are received.

A **theta join** implementation requires all of the data from one stream to be loaded into memory before comparisons between the first and second stream can begin. This blocks the pipeline until the entire first stream has been passed completely to the join activity and is available in memory.

Table 5.6 shows that SPARQL-DQP, using a mediator/wrapper architecture, follows a hybrid shipping paradigm for data/query transfer. Even though the data scan is always done at the server and then transferred to another node for processing, the possibility may exist in which the data node contains an OGSA-DAI server and the process is done first at the data node.

Table 5.7: Optimisation in client-server Architectures in SPARQL-DQP

	Other Systems	...	SPARQL-DQP
Site selection	?	...	No
Optimisation place	?	...	Client
Optimisation time	?	...	Compile
2-step optimisation	?	...	No
Non Blocking operators	?	...	Yes

Table 5.7 shows at what time systems perform the optimisation. All of them perform the optimisation at compile time and do not modify this query plan later on in processing time.

5. ACCESSING DISTRIBUTED RDF DATA STORES

Table 5.8: Adaptive query optimisations in SPARQL-DQP

	Other Systems	...	SPARQL-DQP
Symmetric hash joins	?	...	No
Eddi	?	...	No
N-ary hash joins	?	...	No

In 5.8 shows that query techniques are not implemented in SPARQL-DQP. Although that, we believe that AQP is one of the ways to improve the query processing times, and the community seems to agree with that opinion since these type of techniques are already being applied by systems like ANAPSID [AVL⁺11], ADERIS [LKMT11] or by [LT10].

Table 5.9: Web-service based optimisations in SPARQL-DQP

	Other Systems	...	SPARQL-DQP
Service Parallelisation	?	...	Yes
Indirect Access	?	...	Yes

As it is shown in Table 5.9, SPARQL-DQP implements service parallelisation, streaming and indirect access. Service parallelisation allows us to submit queries where the processing will be distributed amongst multiple nodes to be executed remotely. Query parallelisation is based on [Gra90]. SPARQL-DQP also implements indirect access or asynchronous web service calls, that allows data to be accessed at the time when it becomes available. This can also be considered a type of non-blocking operator for retrieving data.

5.5 Conclusions

In this section we described our implementation system: SPARQL-DQP and described the implementations for accessing distributed RDF data sources. To do that we extended OGSA-DAI and OGSA-DQP, with new data resources and a new query language respectively. This extension allows SPARQL-DQP to access to the full potential of OGSA-DAI, extensively used in e-Science applications and projects¹. In the next chapter we will compare our system with the others described in Section 2.4.1 and prove the validity of our approach.

¹<http://www.ogsadai.org.uk/applications/index.php>

6

Evaluation

The objective of our evaluation is twofold: first, we aim at showing that we can handle SPARQL queries that comply with the SPARQL 1.1 Federated Extension; second, we wish to prove that the optimisation techniques proposed in Section 4.4.1 actually reduce the time needed to process queries when compared to non-optimised approaches.

6.1 Benchmark Suite Design

There are several benchmarks available for the evaluation of SPARQL query processing systems like the Berlin SPARQL Benchmark [BS09], SP²Bench [MSP10] and Fedbench [SGH⁺11]. Unfortunately for our purposes, the first two are not designed for a distributed environment, while the third one is based on a federated scenario but is not as comprehensive as the Berlin SPARQL Benchmark and SP²Bench. Thus, we decided to divide our evaluation in two complementary evaluations:

- The first evaluation is based on some queries from the life sciences domain, similar to those in [SGH⁺11] but using a base query and increasing its complexity like in [BS09]. These queries are real queries used by Bio2RDF experts.
- For our second evaluation we selected Fedbench [SGH⁺11]. Fedbench proposes three sets of queries for evaluating distributed SPARQL query processing systems:
 - A cross domain set of queries, which distribute queries across widely used SPARQL endpoints such as DBpedia¹ or the LinkedMDB endpoint²;

¹<http://dbpedia.org/sparql>

²<http://data.linkedmdb.org/sparql>

6. EVALUATION

- a life sciences set of queries, which evaluate how systems query one of the largest domain in the LOD cloud;
- a version of the SP2Bench [MSP10] benchmark queries, for evaluating the robustness and performance of RDF data stores, providing a wide range of queries that evaluate most of the SPARQL characteristics.

However, the Fedbench evaluation framework has several limitations. First, their queries do not take into account the SPARQL 1.1 Federation extension. That means that the queries do not contain the `SERVICE` keyword. Instead, the query engines have to identify to which endpoint they should direct each part of those queries. Since our aim is to deal with the SPARQL 1.1 Federation extension, we modified manually these queries in the benchmark by adding the `SERVICE` keyword where necessary. Second, Fedbench does not contain queries with `OPTIONAL` keywords, which are common in open world domains like the Semantic Web. Therefore, for completeness and representativeness we added several queries to this benchmark, completing it. These new queries can be found in Appendix A.3.

6.2 Selection of Systems for the Evaluation

We evaluated SPARQL-DQP along with other systems described in Section 2.4.1. In the two evaluations we compared with a different subsets of these systems. In the first evaluation we compared SPARQL-DQP with ARQ, NetworkedGraphs, and DARQ. The selection of these systems for the first evaluation is based on the availability of them. Both evaluations were done at different times, with several months of difference. At the time of performing this first evaluation, the only systems available were DARQ, NetworkedGraphs and ARQ. RDF::Query was available but in its latest release at that moment it had not available the SPARQL federated extension. SemWIQ was also available but its development had stopped, and its architecture differs from ours, since it is a mediator system that integrates a variety of data sources using RDF while we implement and compare with SPARQL distributed query processors. The other systems ([LT11, AVL⁺11, LKMT11]) were still in a development phase and were not available at the time of performing the first evaluation. Besides, since we queried life remote SPARQL endpoints, the evaluation could not be repeated. Data in these endpoints changed and the same evaluation could not be replicated.

For the second evaluation we selected ARQ, RDF-Query, SPLENDID and FedX for comparing with SPARQL-DQP. We chose a variety of systems based on their availability and the implementation of the different optimisations and their was of accessing remote SPARQL endpoints. We selected ARQ and RDF-Query because they are two of the reference implementations of the SPARQL 1.1 Federation.

They implement all the SPARQL 1.1 federation extension and their results are easily comparable with the results from SPARQL-DQP. We also selected SPLENDID and FedX because of their implemented optimisations and their availability. Both of them implement an SPARQL endpoint selection based on the triple pattern predicates in the query. This allows users not to specify directly to which endpoint they want to query but as drawback they may get different results since these systems do not follow the SPARQL 1.1 Federation Extension semantics. Besides of that, to query a set of endpoints instead of querying a single one may result in an excessive overhead in the network traffic. We left out of this comparison the other systems in the state of the art because of their availability. ANAPSID was available at the end of the writing of this thesis and we could not run the tests on time, while [LT11] was not available. The other systems described in 2 ([LT11, LKMT11]) were not available for testing. The amount of systems that implement the SPARQL 1.1 Federation Extension is constantly growing, specially since the specification is close to finish its Last Call document status. We can not cover all of them, thus we selected a significant representation of those systems (some of the most cited and downloaded).

We chose these systems because two of them are part of the official SPARQL implementations and SPLENDID is based in Sesame and adds statistics models for join reordering. In general, with large amounts of data they do not perform well while when data is fewer all systems are more equal. Since SPLENDID does not implement the official SPARQL Federation extension generates its query plans from the statistics about each triple in the query. Its performance is very good when the endpoints to query are the same than the ones pointed by the SERVICE keyword. But if there are more endpoint descriptions its performance is worse, but it gets more complete results. ARQ executes the queries in a different way, trying always to obtain all results from the remote SPARQL endpoints, which generates many connection error from the servers. RDF-Query is the most "standard" query engine, and performs good in queries that do not deal with large amounts of data.

6.3 Evaluation 1

In the first evaluation, we compare the results and performance of our system with the selected systems that provide some support for SPARQL query federation. The objective of this evaluation is to show first that we can handle SPARQL queries that comply with the federated extension, and second that the optimisation techniques proposed in Chapter 4 actually reduce the time needed to process queries. We decided to base our evaluation on some queries from the life sciences domain, similar to those in [SGH⁺11] but using a base query and increasing its complexity like in [BS09]. These queries are real

6. EVALUATION

queries used by Bio2RDF experts. Our evaluation was done in an Amazon EC2 instance. The instance has 2 cores and 7.5 GB of memory run by Ubuntu 10.04.

6.3.1 Data set description

The Bio2RDF data sets contains 2,3 billion triples organised around 40 data sets with sometimes overlapping information. The Bio2RDF data sets that we have used in our benchmark are: Entrez Gene (13 million triples, stored in the local endpoint sparql-pubmed), Pubmed (797 million triples), HHPID (244,021 triples), and MeSH (689,542 triples, stored in the local endpoint sparql-mesh). Table 6.1 summarises this endpoints description. One of the practical problems that these benchmarks have is that public SPARQL endpoints normally restrict the amount of results that they provide. To overcome this limitation we installed Entrez Gene and MeSH in servers without these restrictions. We also divided them in files of 300,000 triples (even tough our system is capable of managing millions of results from the remote SPARQL endpoints), creating endpoints for each one of them.

	Triples	Out Links	In links	Namespace	Limit
EntrezGene	1,327,212	-	-	http://bio2rdf.org/	10,000
Pubmed	797,000,000	100,000	21,289	http://bio2rdf.org/	300,000
MeSH	689,542	1,230	-	http://bio2rdf.org/	-
HHPID	244,021	-	-	http://bio2rdf.org/	-

Table 6.1: Cross Domain Endpoint Characteristics

6.3.2 Queries used in the evaluation

Our goal for this evaluation was to show how systems scale with large amounts of data and how well the rewriting rules identified in Section 4.4.1 performed. For that we used 7 queries in our evaluation of increasing order of complexity. In the first query, we just query two remote SPARQL endpoints, for showing how well the systems perform with a a basic join query. Next, we added an OPTIONAL operator to the same query, for showing how well they perform with this new operator. The next queries work in a similar way: we keep adding queries to remote SPARQL endpoints, first with a join and next with an OPTIONAL operator substituting the last join operation. The last query puts one of the SERVICE patterns inside the OPTIONAL pattern. The query structure follows the following path: using the Pubmed references obtained from the Entrez gene data set, we access the Pubmed endpoint (queries Q1 and Q2). In these queries, we retrieve information about genes and their references in the Pubmed data

set. From Pubmed we access the information in the National Library of Medicine’s controlled vocabulary thesaurus (queries Q3 and Q4), stored at MeSH endpoint, so we have more complete information about such genes. Finally, to increase the data retrieved by our queries we also access the HHPID endpoint (queries Q5, Q6 and Q7), which is the knowledge base for the HIV-1 protein. Next we show query Q4 to give the reader an idea of the type of queries that we are considering (a complete list of the queries used can be found in Appendix A.2.1):

```
SELECT ?pubmed ?gene1 ?mesh ?descriptor ?meshReference
WHERE
{
  SERVICE <http://pubmed.bio2rdf.org/sparql> {
    ?pubmed <http://bio2rdf.org/pubmed_resource:meshref> ?mesh .
    ?mesh <http://bio2rdf.org/pubmed_resource:descriptor> ?descriptor .}
  OPTIONAL {
    SERVICE <http://127.0.0.1:2021/sparql-mesh> {
      ?meshReference <http://www.w3.org/2002/07/owl#sameAs> ?descriptor .}
    }
  SERVICE <http://127.0.0.1:2020/sparql-pubmed> {
    ?gene1 <http://bio2rdf.org/geneid_resource:pubmed_xref> ?pubmed .}
}
```

The following table summarises the characteristics of each query, presenting the number of joins in the query, number of OPTIONAL, UNION and FILTER patterns, variables used, constants and solution modifiers. Each query uses as a constant the predicates of the queries.

	Joins	OPT	FILTER	UNION	Patterns	Variables	Solution Modifiers	Bounded Subjects	Bounded Predicates	Bounded Objects
Q1	1	0	0	0	2	4	0	0	3	0
Q2	0	1	0	0	2	4	0	0	3	0
Q3	2	0	0	0	3	5	0	0	4	0
Q4	1	2	0	0	3	5	0	0	4	0
Q5	3	0	0	0	4	5	0	0	5	0
Q6	2	1	0	0	4	5	0	0	5	0
Q7	2	1	0	0	4	6	0	0	5	0

6.3.3 Results

The results of our evaluation are shown in the following table:

Table 6.2 show the results for our first evaluation¹. A first clear advantage of our implementation is the ability to use asynchronous calls facilitated by the use of indirect access mode, what means that we do not get time out in any of the queries. This time out happens when accessing an online distributed

¹We represent as 10+ min. those queries that need more than 10 minutes to be answered.

6. EVALUATION

Query	Not optimised SPARQL-DQP	Optimised SPARQL-DQP	DARQ	NetworkedGraphs	ARQ
Q1	79,000ms.	79,000ms.	10+ min.	10+ min.	440,296ms.
Q2	64,179ms.	64,179ms.	10+ min.	10+ min.	10+ min.
Q3	134,324ms.	134,324ms.	10+ min.	10+ min.	10+ min.
Q4	152,559ms.	136,482ms.	10+ min.	10+ min.	10+ min.
Q5	146,575ms.	146,575ms.	10+ min.	10+ min.	10+ min.
Q6	322,792ms.	79,178ms.	10+ min.	10+ min.	10+ min.
Q7	350,554ms.	83,153ms.	10+ min.	10+ min.	10+ min.

Table 6.2: Evaluation 1 Query Results

query processing like in the case of ARQ. It is important to note that the ability to handle this type of queries is essential for many types of data-intensive applications, such as those based on Bio2RDF. Data transfer also plays a key role in query response times. For example, in some queries the local query engine received 150,000 results from Entrez gene, 10,000 results from Pubmed, 23,841 results from MeSH and 10,000 results from HHPID. The implemented optimisations are less noticeable when the amount of transferred data is fewer.

It is possible to observe three different sets of results from this preliminary evaluation. The first set (Q1–Q3 and Q5) are those that are not optimised because the reordering rules in Section 4.4.1 are not applicable. The second query group (Q4) represents the class of queries that can be optimised using our approach, but where the difference is not too relevant, because of the less amount of transferred data. The last group of queries (Q6–Q7) shows a clear optimisation when using the well-designed patterns rewriting rules. For example, in query 6 the amount of transferred data varies from a join of $150,000 \times 10,000$ tuples to a join of $10,000 \times 23,841$ tuples (using Entrez, Pubmed and MeSH endpoints), which highly reduces the global processing time of the query. Regarding the comparison with other systems, they do not properly handle these amounts of data.

6.4 Evaluation 2

In this section we describe the second evaluation performed. In this evaluation we use the set of queries defined in [SGH⁺11] extending it with a new set of queries that cover a wider range of SPARQL patterns. These queries access three different sets of SPARQL endpoints: a cross domain set of endpoints (DBPedia, New York Times endpoint, LinkedMDB endpoint, Geo endpoint, El Viajero endpoint and

the CIA World Factbook endpoint), a life sciences set of endpoints (Drugbank, Kegg) and a modified set of queries from the SP2Bench [MSP10] SPARQL 1.0 benchmark.

6.4.1 Data sets description

For the cross domain queries we used the data sets available at the DBpedia, LinkedMDB, Geonames and the New York Times endpoints, adding queries to El Viajero SPARQL endpoint. We do not download any data to a local server, instead we query directly these endpoints so we show the real performance of the evaluated systems in real conditions. A summary of the characteristics of these endpoints is showed in 6.3. We performed in the same way for the life sciences queries, accessing the default SPARQL endpoints also providing a summary in 6.4. For the SP2Bench queries we generated a data set of 1.000.000 triples which we clustered into 5 different SPARQL endpoints in a local server. The local SPARQL endpoints were Journal (410.000 triples), InCollections (8.700 triples), InProceedings (400.000 triples), People (170.000 triples), Masters (5.600 triples) and PhDs (5.900 triples). Table 6.5 summarises these SPARQL endpoints characteristics.

	Triples	Out Links	In links	Namespace	Limit
DBpedia	1,000,000,000	16,480,998	11,531,095	http://dbpedia.org/resource/	1000
NYTimes	345,889	23,700	21,289	http://data.nytimes.com/	1000
LinkedMDB	6,148,121	162,756	1,883	http://data.linkedmdb.org/resource/	1000
Geonames	93,896,732	-	1,028,252	http://sws.geonames.org/	1000
World Factbook	38,640	-	665,005	http://www4.wiwiw.fu-berlin.de/factbook/resource/	1000
El Viajero	9,462,339	12,750	-	http://webenemasuno.linkeddata.es/elviajero/resource/	1000

Table 6.3: Cross Domain Endpoint Characteristics

	Triples	Out Links	In links	Namespace	Limit
DBpedia	1,000,000,000	16,480,998	11,531,095	http://dbpedia.org/resource/	1000
Kegg	50,811,634	5,242,739	6,031	http://bio2rdf.org/	1000
Drugbank	765,936	47,857	30,148	http://www4.wiwiw.fu-berlin.de/drugbank/resource/	1000

Table 6.4: Life Sciences Endpoint Characteristics

6.4.2 Queries used in the evaluation

In the next lines we describe the queries used in our evaluation. The SPARQL code of these queries can be found in Appendix A.3. The original Cross Domain query set did not contain any query that

6. EVALUATION

	Triples	Out Links	In links	Namespace	Limit
Journal	-	-	-	http://localhost/publications/articles/	-
InProceedings	-	-	-	http://localhost/publications/inprocs/	-
People	-	-	-	http://localhost/persons/	-
InCollections	-	-	-	http://localhost/publications/incolls/	-
Master's Thesis	-	-	-	http://localhost/publications/masters/	-
PhD Thesis	-	-	-	http://localhost/publications/phds/	-

Table 6.5: Life Sciences Endpoint Characteristics

used nor the OPTIONAL neither the FILTER operators. Thus, we decided to complement the query set with queries 4b, 8 and 9. The purpose of these queries was to show how systems behave with the OPTIONAL and FILTER operators, and also a combination of the both. Next, we describe the set of queries that access a cross domain set of SPARQL endpoints (the SPARQL code of these queries can be found in Appendix A.3.1):

Fedbench Cross domain Query 1 queries DBPedia and the NYTimes endpoints. DBpedia is asked for the properties and values of the resource that represents Barack Obama. Then, the results are unioned with the results of the query to the NYTimes, which asks for the news in which resources with link to Barack Obama in the DBPedia appeared in any news.

Fedbench Cross domain Query 2 queries again DBPedia and the NYTimes SPARQL endpoints. This query asks for Barack Obama's political party in the DBPedia endpoint and also, in the NYTimes endpoint the pages in which Barack Obama appeared.

Fedbench Cross domain Query 3 is similar to the cross domain query 2, but issuing a more complex query to the DBPedia endpoint, which this time specifies the type of the resource Barack Obama

Fedbench Cross domain Query 4 queries the LinkedMDB movie database and the NYTimes SPARQL endpoint. The query to the LinkedMDB SPARQL endpoint asks for the actors that participated in the "Tarzan" movie, which are joined later on with the news that any of these actors appeared in the NYTimes.

Fedbench Cross domain Query 4b extends the previous query by adding a FILTER out of the SERVICE operators. Semantically is almost an equivalent query to the previous one, but adding a filter instead of searching for a specific value.

Fedbench Cross domain Query 5 queries again the LinkedMDB and the DBPedia endpoints. The query issued to the DBPedia endpoint asks for directors whose nationality is Italy while the LinkedMDB query asks for movies with genre. Results from both queries are joined.

Fedbench Cross domain Query 6 queries the LinkedGeoData endpoint for artists that are based in Germany.

Fedbench Cross domain Query 7 queries again the LinkedGeoData endpoint for places in "California" and the NYTimes endpoint for any news, for later on joining them.

Due to the poor query diversity in the Fedbench query set we added two more queries:

Fedbench Cross domain Query 8 The first one queries DBPedia for countries and their area, for optionally obtaining from the El Viajero endpoint all the travel guides associated to these countries. We filter these results to those countries that have a land smaller than 20000 km.

Fedbench Cross domain Query 9 extends the previous one by adding data obtained from the CIA World Fact book endpoint instead of using FILTER.

The following tables summarizes the main characteristics of each query in the cross domain query set:

	Joins	OPT	FILTER	UNION	Patterns	Variables	Solution Modifiers	Bounded Subjects	Bounded Predicates	Bounded Objects
Q1	0	0	0	1	2	2	0	1	1	1
Q2	1	0	0	0	2	3	0	1	3	1
Q3	1	0	0	0	2	3	0	0	5	1
Q4	1	0	0	0	2	2	0	0	5	1
Q4b	0	1	1	0	3	5	0	0	6	1
Q5	1	0	0	0	2	4	0	0	4	1
Q6	0	0	0	0	1	3	0	0	4	1
Q7	1	0	0	0	2	2	0	0	4	1
Q8	0	1	1	0	2	3	0	0	3	1
Q9	1	1	0	0	3	4	0	0	7	2

The life sciences domain queries contain a variety of SPARQL queries, including OPTIONAL, FILTER and UNION operators, thus, we decided not to add any new query to the existing ones. can be found in Appendix A.3.2 and a brief description of those queries is presented in the following lines:

Fedbench Life Sciences query 1 unions the results from querying DBpedia first for drugs and their melting points and the same values in DBPedia.

6. EVALUATION

Fedbench Life Sciences query 2 unions the data of a specific resource in Drugbank with the same data in other SPARQL endpoint.

Fedbench Life Sciences query 3 joins two queries, one to DBPedia asking for drugs and other to Drugbank asking for interactions of the same drug.

Fedbench Life Sciences query 4 joins the results of a query to Drugbank asking drug categories and their compounds for next joining them with the results to the Kegg SPARQL endpoint, which asks for equations for these compounds.

Fedbench Life Sciences query 5 queries Kegg and Drugbank again. This time Kegg is queried for drugs and their images. The resulting data is next joined with the data about these drugs in Drugbank.

Fedbench Life Sciences query 6 obtains data about drug’s micro-nutrients (from Drugbank) and then joins these data with the same type of data from Kegg.

Fedbench Life Sciences query 7 queries first for drugs that affect humans or mammals in Drugbank, for later on joining the results of these drugs with the same drugs in Kegg but filtering those drugs with mass greater than 5. Next, we optionally add the bio transformation information from Drugbank again.

The following table summarises the characteristics of the life science queries:

	Joins	OPT	FILTER	UNION	Patterns	Variables	Solution Modifiers	Bounded Subjects	Bounded Predicates	Bounded Objects
Q1	0	0	0	1	2	2	0	0	2	0
Q2	0	0	0	1	2	2	0	2	0	0
Q3	1	0	0	0	2	3	0	0	5	1
Q4	2	0	0	0	2	3	0	0	6	0
Q5	2	0	0	0	2	3	0	0	6	1
Q6	2	1	0	0	2	2	0	0	5	1
Q7	0	1	0	0	2	3	0	0	3	0

The proposed SP2Bench queries by Fedbench did not contain much OPTIONAL nor FILTER patterns, thus, we decided to add more queries with these operators. The new queries are 7b, 7c, 8b, 8c and 8d, which contain the combination of OPTIONAL and FILTER patterns with the existing JOIN patterns. The queries based on the SP2Bench benchmark are described in the next lines (the SPARQL code corresponding to these queries is in [Appendix A.3.3](#)):

Fedbench SP2Bench query 1 asks to the Journal endpoint for those journals with title "Journal 1 (1940)"

Fedbench SP2Bench query 2 asks to the InProceedings endpoint for authors, abstracts and book titles.

Fedbench SP2Bench query 3 asks again the InProceedings endpoint for all article with information about their number of pages.

Fedbench SP2Bench query 4 is very similar to the previous query but querying the Journal endpoint and the property month.

Fedbench SP2Bench query 5 is also similar to the previous two, but querying the InProceedings endpoint for articles that have ISBN number.

Fedbench SP2Bench query 6 is a more complex query for remote endpoints. It queries the Journal endpoint twice for authors and journals in which those authors published something with a query to the People endpoint that asks information for these authors. This is a very costly query for remote SPARQL endpoints.

Fedbench SP2Bench query 7 Here we query three SPARQL endpoints. First we query the Journal endpoint for journal articles and their authors resource id. Next, we query the InProceedings endpoint for those people that published something in a conference, joining the people results with the previous query results. Next, we ask for extra information to the people endpoint.

Fedbench SP2Bench query 7b adds to the previous query more complexity adding an OPTIONAL operator. We query for people who published in a journal and optionally obtaining the conferences in which these people also published. Next, we obtain the people's information.

Fedbench SP2Bench query 7c modifies query 7, adding to it a FILTER operator. Now we query the Journal endpoint for journals and their authors and optionally obtaining the author's names. We filter journal papers to those with 200 pages.

Fedbench SP2Bench query 8 follows the same structure than in the query before, but this time querying the InCollections endpoint instead of the Journal endpoint. The InCollections endpoint contains much less data than the Journal endpoint.

Fedbench SP2Bench query 8b is a variation of the previous query number 8. In there we add the OPTIONAL keyword. We query for people and optionally their papers in conferences. We finally

6. EVALUATION

join the results with the results of a query to the InCollection endpoint. This query allows us to reduce the amount of intermediate results between operators.

Fedbench SP2Bench query 8c adds another SPARQL endpoint, accessing four SPARQL endpoints.

The query structure is as follows: First, we query for conference papers in the InProceedings endpoint. We optionally obtain the author’s names. We join all these results with the journal papers in which conference authors also appear. We finally join the results with the results obtained from querying the InCollections endpoint authors and papers. This query is the most complex one since it access four different endpoints, and data transfer is more noticeable. Here, since processing and data transfer is high, the optimisations described in 4 provide a great time improvement.

Fedbench SP2Bench query 8d asks for conference papers and optionally obtaining the authors names.

We filter by year. In this query the optimisations are also noticeable.

The following table summarises the characteristics of the SP2Bench queries:

	Joins	OPT	FILTER	UNION	Patterns	Variables	Solution Modifiers	Bounded Subjects	Bounded Predicates	Bounded Objects
Q1	0	0	0	0	1	1	0	0	3	1
Q2	0	0	0	0	1	10	0	0	10	0
Q3	0	0	0	0	1	1	0	0	1	1
Q4	0	0	0	0	1	1	0	0	1	1
Q5	0	0	0	0	1	1	0	0	1	1
Q6	1	0	0	0	2	2	1	0	8	2
Q7	2	0	0	0	3	2	1	0	5	1
Q7b	1	1	0	0	3	4	1	0	5	1
Q7c	0	1	1	0	3	6	1	0	4	1
Q8	2	0	0	0	3	4	1	0	5	1
Q8b	1	1	0	0	3	4	1	0	5	1
Q8c	2	1	1	0	4	5	1	0	16	3
Q8d	0	1	1	0	3	2	1	0	5	1

6.4.2.1 New queries used in the evaluation

We added three queries to the cross domain query set and five more queries to the SP2Bench set of queries. The new cross domain queries are query cd4b.rq, cd8.rq and cd9.rq. In cd4b.rq we added a new FILTER to the original query (cd4.rq), asking now for those actors that appear in any NY Times news, filtering for the film 'Tarzan'. In this query we apply rule number 1 from the optimisation list. Next, we add two completely new queries. In the first one (cd8.rq) we query DBPedia and the Elviajero[GVC11]

SPARQL endpoint¹ for data about countries and existing travel books, we filter for countries with land grater than 20000km2, using rule 1 as well. Lastly we added cd9.rq in which we query DBPedia for countries and optionally we get the existing travel books for these countries from the El Viajero endpoint, completing this information with the climate data at the CIA world factbook SPARQL endpoint². Next we show two of the cross domain queries to give the reader an idea of the type of queries that we are considering:

```
SELECT ?Book ?Country ?Area
WHERE {
  SERVICE <http://dbpedia.org/sparql> {
    ?Country rdf:type dbonto:Country .
    ?Country dbprop:areaKm ?Area
  }
  OPTIONAL {
    SERVICE <http://webenemasuno.linkeddata.es/sparql> {
      ?Book viajero:refersTo ?Country .
    }
  }
  FILTER(?Area < "20000"^^xsd:integer)
```

```
SELECT ?Book ?Country ?Area ?climate
WHERE {
  SERVICE <http://dbpedia.org/sparql> {
    ?Country rdf:type dbonto:Country .
    ?Country dbprop:areaKm ?Area .
    ?Country rdfs:label ?countryLabel
  }
  OPTIONAL {
    SERVICE <http://webenemasuno.linkeddata.es/sparql> {
      ?Book viajero:refersTo ?Country .
    }
  }
  SERVICE <http://www4.wiwiw.fu-berlin.de/factbook/sparql> {
    ?CountryCIA rdfs:label ?countryLabel .
    ?CountryCIA rdf:type factbook:Country .
    ?CountryCIA factbook:climate ?climate
  }
}
```

We also added five queries to the SP2Bench set of queries in Fedbench. These queries are an extension of the SP2Bench queries 7 and 8 which ask for proceedings or journals and their authors. We added an extra level of complexity first by adding `OPTIONAL` to those queries. Query 7b asks for journals, optionally it obtains the authors' publications in a conference, and later it obtains also the authors' names. Query 8b asks for all people and optionally obtains all the papers these people published in a conference. It also asks for all in collections in which these people appear as authors. In query7c.rq we ask for all journals, obtaining their authors with an optional and filtering for the number of pages. Query8c.rq is the most complex query since it queries 4 different SPARQL endpoints. In this query

¹<http://webenemasuno.linkeddata.es/sparql>

²<http://www4.wiwiw.fu-berlin.de/factbook/sparql>

6. EVALUATION

we query for all InProceedings, optionally obtaining the people who wrote them, next asking for those authors that also wrote a journal paper and also an in collections. Query8d.rq asks for all the in proceedings, optionally obtaining their authors and limiting the output data to those proceedings from the year 1950.

6.4.3 Results

The evaluation was carried out in an Intel XEON machine with 4 cores and 8GB of memory. The database setup used in the SP2Bench evaluation queries has been the following: for storing the RDF data we used a Jena TDB¹ triple storage and as the SPARQL endpoint server we used Jena's Fuseki² server.

We run each query twice for warming up the systems evaluated. Next, we run five times each query and calculate the geometric mean of each query execution.

The results of our evaluation are shown in Table 6.8 for the SP2Bench set of queries and Table 6.6 for the Cross Domain set of queries. For the life sciences set of queries we refer to Table 6.7 and [BAAC11].

Query	Not optimised SPARQL-DQP	Optimised SPARQL-DQP	ARQ	RDF::Query	SPLENDID	FedX
Q1	2,117ms.	2,150ms.	2,514ms.	2,144ms.	18,000ms.	1,174ms
Q2	2,370ms.	2,200ms.	2,470ms.	2,113ms.	5,000ms.	333ms.
Q3	9,659ms.	8,085ms.	16,789ms.	18,920ms.	2,000ms.	357ms.
Q4	8,341ms.	7,291ms.	1,073ms.	19,203ms.	1,000ms.	161ms.
Q4b	10,562ms.	9,771ms.	server error.	1,154ms.	90,000ms.	431ms.
Q5	5,687ms.	3,759ms.	16,933ms.	7,815ms.	34,000ms.	1ms.
Q6	887ms.	0,878ms.	0,674ms.	0,783ms.	1,000ms.	0.001ms.
Q7	11,625ms.	7,994ms.	0,336ms.	20,680ms.	1,000ms.	1ms.
Q8	15,544ms.	14,373ms.	24,306ms.	1,916ms.	skips OPTIONAL evaluation.	8,083ms.
Q9	19,658ms.	17,192ms.	10+ min.	10+ min.	skips OPTIONAL evaluation.	1Ms.

Table 6.6: Results of Cross domain queries.

The results from Table 6.6 show how the systems evaluated behave in a typical situation, in which users query some of the most common SPARQL endpoints. The data returned by each endpoint is at most of 10.000 results (from the El viajero endpoint) where the usual result size is of 2.000 results. This

¹<http://openjena.org/TDB/>

²<http://openjena.org/wiki/Fuseki>

Query	Not optimised SPARQL-DQP	Optimised SPARQL-DQP	ARQ	RDF::Query	SPLENDID	FedX
Q1	5,601ms.	5,333ms.	10,362ms.	5,109ms.	3,000ms.	17,052ms.
Q2	2,316ms.	2,299ms.	2,753ms.	2,136ms.	70,000ms.	2,318ms.
Q3	83,728ms.	89,112ms.	server error.	89,202ms.	42,000ms.	server error
Q4	12,708ms.	27,735ms.	98,716ms.	55,829ms.	4,000ms.	333ms.
Q5	6,021ms.	5,811ms.	server error.	558ms.	4,000ms.	22,606ms.
Q6	8,221ms.	9,077ms.	14,358.	37,203ms.	180,000ms.	1,516ms.
Q7	10,493.	10,918ms.	server error.	56,6204ms.	skips OPTIONAL evaluation.	24,384ms.
Q8	85,799ms.	86,329ms.	server error.	88,111ms.	10min.	3ms.

Table 6.7: Results of life sciences domain queries.

makes all systems to answer the queries in reasonable times. FedX outruns all the other systems in all the queries. This demonstrates that their approach is better for these type of user queries. The queries in which not so many results are returned, FedX optimisation techniques are used efficiently (specially the BINDINGS optimisation). Queries in which times are of 0.001 for FedX mean that they did not find any result in the query. Our system performs similar to the other systems but in Query 8, in which RDF::Query outperforms all systems. We believe that this is because that systems inserts the FILTER inside the SERVICE invocation. When the amount of data returned by the remote SPARQL endpoints increases, our system performs better as Query 9 shows.

Results from Table 6.7 also show a similar behaviour in the Life Science domain. In this domain we did not add any extra query to the evaluation, since a more complete evaluation can be found in [BAAC11]. FedX well in some queries with several differences. In the life science endpoints more predicates used in the queries are repeated, thus, the same queries are sent to several endpoints (due to FedX and SPLENDID use triple pattern predicates to identify to which endpoint route queries), differing from those systems that using the SPARQL 1.1 Federation Extension only send queries to one SPARQL endpoint. Besides, life sciences servers behaved a bit worse (in our evaluation) returning server errors, specially when BINDINGS queries were sent. Regarding SPARQL-DQP and the other systems, they performed similarly but when the data returned by the remote SPARQL endpoints is increased. In that situation, SPARQL-DQP worked better than other systems. The implemented optimisations (specially the implementation of the pattern reordering rules described in Section 4.4.1) are less noticeable when the amount of transferred data is fewer.

The results from Table 6.8 show how the evaluated systems behave with large amounts of data. None of the SPARQL endpoints have a limit in the number of the returned results. The People endpoint

6. EVALUATION

Query	Not optimised SPARQL-DQP	Optimised SPARQL-DQP	ARQ	RDF::Query	SPLENDID	FedX
Q1	275ms.	281s.	6ms.	0,106ms.	440,296ms.	784ms.
Q2	26,862ms.	26,800s.	27,550ms.	77,327ms.	10+ min.	78,416ms.
Q3	268ms.	272s.	14ms.	119ms.	10+ min.	10+ min.
Q4	270ms.	300s.	78ms.	305,413ms.	10+ min.	10+ min.
Q5	264.	269s	13ms.	115ms.	10+ min.	10+ min.
Q6	Time Out.	Time Out	Time out.	Time out.	10+ min.	10+ min.
Q7	37,421ms.	37,581s	10+ min.	45,995ms.	10+ min.	10+ min.
Q7b	41,399ms.	41,620s	10+ min.	10+ min.	10+ min.	10+ min.
Q7c	44,566ms.	41,392s	10+ min.	10+ min.	10+ min.	10+ min.
Q8	23,597ms.	21,776s.	10+ min.	10+ min.	10+ min.	10+ min.
Q8b	24,306ms.	25,005s.	10+ min.	10+ min.	10+ min.	10+ min.
Q8c	452,479ms.	46,009s.	10+ min.	10+ min.	10+ min.	10+ min.
Q8e	43,193ms.	39,901s.	10+ min.	10+ min.	10+ min.	10+ min.

Table 6.8: Results of the SP2B evaluation.

contains 82.685 persons with name, the InProceedings endpoint contains 65.863 in proceedings with author, the InCollection contains 615 in collection with author and the Journal endpoint contains 83.706 journals with author. In this set of queries and with these data our system performs similar to the others in the first five, even a bit worse. This is due to the Web services invocation that our system is doing, therefore, for a small number of results our system is easily beaten. However, when the result number is larger, our system is able to return results in reasonable times while the other systems need much more time to return results, when they return results at all. Our architecture allows us to deal with large amounts of data in a robust and efficient way. FedX which outperformed the other systems in the other evaluations, this time performs badly, specially because all endpoints contain similar predicates, and the same queries are sent to all of them. Besides, the use of BINDINGS with a large amount of results generates too many sub queries that generate a high overload in the remote system.

It is possible to observe three different sets of results from this evaluation. The first set (standard Fedbench Life Sciences domain, Cross domain and SP2 queries) are those that are not optimised because the reordering rules in Section 4.4.1 are not applicable. The second query group represents the class of queries that can be optimised using our approach, but where the difference is not too relevant, because the less amount of transferred data. In this query group we identify query 7 in the Life Sciences domain, Q4 in [BAAC11], queries 4b, 8 and 9 in the cross domain query set and queries 7b, 7c and 8b. It is possible to observe a reduction of the query response times in most of these queries, specially when

using rule 1. The last group of queries (Q6–Q7 in the evaluation at [BAAC11] and queries 8c and 8d of the SP2bench queries) shows a clear optimisation when using the well-designed patterns rewriting rules. For example, in query 6 the amount of transferred data varies from a join of $150,000 \times 10,000$ tuples to a join of $10,000 \times 23,841$ tuples (using Entrez, Pubmed and MeSH endpoints), which highly reduces the global processing time of the query. properly handle these amounts of data. We represent as 10+ min. those queries that need more than 10 minutes to be answered. In query sp2b8c.rq the reduction of intermediate results is done by joining the SERVICE call to the InProceedings endpoint (which queries a subset of the existing proceedings) and the results of a query to the Journal endpoint. The amount of results of this join is 830 results instead of the 32770 results of performing first an optional. Next, we join the results with the existing InCollection, which are 615 results. This reduces one order of magnitude the query response time.

As commented before, not to identify previously the endpoint where to route the query impacts the results and this is reflected in the comparison tables. In the SP2Bench queries, it is highly noticeable that these systems (FedX and SPLENDID) fail to answer most of the queries since they try to access too many SPARQL endpoints due that they have the same predicates. The SP2Bench dataset is a single RDF file divided into several SPARQL endpoints. Thus, the comparison between other systems may not be that fair. Besides, to query several SPARQL endpoints sharing common predicates may cause that one of these queries return a small amount of results. Thus, the join with the other predicates may result in a smaller amount of results. This, as note before, is not compliant with the official SPARQL 1.1 Federation Extension, and again, the results and the comparison with the system that follow this specification may not match.

The most complex query executed in the second evaluation has been query 8c. This query accessed four different SPARQL endpoints and contained two joins and one OPTIONAL operation. The execution of this query without optimisation requires much more time to produce results than the non optimised one. The non optimised query first executes remote SPARQL query for next, adding more data to the existing results via an OPTIONAL operator. Thus, we place a query in which 10 results are returned for next adding one more result. This sub query requires processing from the remote SPARQL endpoints first (for obtaining the results, one of these queries retrieves all data from one of them) and processing for joining all the data. Next, we join all these data with also a generic query which returns most of the papers in the Articles SPARQL endpoint, for next performing another join with a smaller query to the data in the InCollections endpoint. When this query is optimised, first, rule 2 is applied (see Section 4.4.1) placing first a join between a Journal query and the InProceedings endpoint. This means

6. EVALUATION

a left join of 830 results instead of the 32770 results of performing first an optional. Similar results can be seen in evaluation 1 (Table [6.2](#)), in which queries with 4 SERVICE patterns performed similarly.

7

Conclusions and future work

This thesis presents several contributions to the state of the art that aim to address several research problems in the area of distributed SPARQL query processing. These problems and our contributions to their solution are summarised below.

7.1 Conclusions

Querying distributed SPARQL data sets can be a very time-costly task. Users send queries which need to access different sets of RDF data sets for which an optimal query plan has to be generated. Current approaches focus on querying the remote SPARQL endpoints directly, providing a small amount of optimisation techniques. While this may be sufficient for small numbers of results, systems clearly turn to be too slow when dealing with large quantities of data. In our evaluation, all the systems (but ours) needed more than 10 minutes for returning results to the user's queries in the scenario in which large amounts of data were involved. Up until now, a formal study of the semantics of the SPARQL 1.1 Federation Extension, which allows SPARQL queries to be redirected to a set of endpoints, has been missing.

Thus, the first objective of this thesis focused on **providing such a formal study of the SPARQL 1.1 Federation extension**. This was done in Chapter 4, where we identified associated problems with the current version of the specification document. The main issue identified was the lack of well-defined semantics for specific SPARQL patterns, for example the `SERVICE ?X` pattern. Besides, the evaluation of the `SERVICE ?X` pattern also yields problems when an evaluation order for the SPARQL query has not been included. We provided formal semantics for the SPARQL 1.1 Federation Extension along with a suggestion for an evaluation order for such queries.

7. CONCLUSIONS AND FUTURE WORK

The accomplishment of the second objective of the thesis (**to propose optimisation techniques for the evaluation of SPARQL queries in distributed settings**) has also produced significant contributions to the state of the art as reflected in [BAAC11]. By formalising the semantics of the SPARQL 1.1 Federated Extension we identified a set of query rewriting techniques for query optimisation. These techniques were originally proposed in [PAG09] for SPARQL 1.0 without the distributed component in the SPARQL query. We also implemented the rewriting rules for optimising SPARQL 1.1 Federated queries, and showed that it produces good performance. The accomplishment of this second objective also validates hypothesis 2, **it is possible to optimise distributed queries using pattern reordering without degrading performance and improving it for the OPTIONAL and FILTER SPARQL patterns**.

The second methodological objective was related to technological objectives: **to develop a tool for querying distributed RDF data sets using the SPARQL query language - SPARQL-DQP and to implement optimisation techniques for accessing distributed RDF data identified in the methodological objectives**. We implemented the SPARQL-DQP system which allows users to execute federated queries to a set of SPARQL endpoints using a single query. SPARQL-DQP implements the SPARQL 1.1 Federation extension and the optimisation techniques described in methodological objective 2. In our implementation we extended the OGSA-DAI/DQP framework for accessing data with a new type of OGSA-DAI data resource (RDF data resource) and added support for a new query language (SPARQL). The implementation uses the SQL operators already existing in this system, plus new specific operators like the SPARQL UNION operator or specific user defined functions for filtering query results. This extension using SQL operators illustrates Hypothesis 1: **Work done in querying distributed relational databases can be applied to querying distributed RDF data**. Furthermore, the existing SQL optimisers in OGSA-DQP are also utilised for SPARQL. For applying such optimisations it is necessary first to perform an analysis of the semantics of SPARQL due to its differences for reordering operators. An example are the semantics of the SPARQL OPTIONAL operator whose reordering, due to the different interpretation of null values between SQL and SPARQL, would produce different results. Hypothesis 3, **(existing tools for accessing distributed data sources specifically tailored for large amounts of data are more suitable than existing approaches for querying distributed RDF data)** is also demonstrated by extending OGSA-DAI/DQP with our system.

The third methodological objective (**to propose a test suite for measuring the performance of the different Distributed SPARQL Query Processing systems**) was accomplished by the proposal of the set of evaluations described in 6. Existing frameworks for benchmarking SPARQL query applications, like [SGH⁺11, MSP10, BS09] do not provide a complete set of comprehensive queries for analysing

the performance of distributed SPARQL query processing systems, missing for example more patterns with the OPTIONAL keyword or more data queries. They lack some types of queries (missing a variety of query patterns - for example several combinations of OPTIONAL) plus the ability to work in a decentralised environment. We provide an extension to these evaluation frameworks by adding a greater variety of queries, specifically adding FILTER and OPTIONAL patterns. We also extend it by adding queries to more life science data sets and combining remote and local RDF data stores. Thus, we provide a more complete test suite which allows a more detailed evaluation of the distributed SPARQL query processing systems.

7.2 Future work

In this section we describe those research problems that were not tackled in this thesis, either for being out of the scope of this thesis or due to time-permitting questions.

As for future work, one of the main contributions of this PhD thesis has been the formulation of the mechanisms to reorder SPARQL queries that access distributed RDF data sources. This reordering is achieved by using the special characteristic of the OPTIONAL operator, which has the same semantics as the Left Outer Join SQL operator. The difference with SQL is that SPARQL treats null values just as unbound values. Thus, the work relating to the left outer join reordering described in [GLR97] could be applied here as well as the main restriction identified in [GLR97] are null values.

One of the main problems we faced during this work was the difficulty in obtaining statistics about the different SPARQL endpoints we queried. Currently there are initiatives for describing statistics about the data stored like VoID¹, but many public SPARQL endpoints do not yet make available this type of statistics. Therefore, for optimising distributed queries it will be necessary to find another approach. As commented in the previous paragraph, reordering without using statistics is an option. Another one might be to use adaptive query processing techniques. Using adaptive query processing it is possible to adapt the query processing to take into account network changes, like latency or unavailability of the data sources. Existing work like that proposed in [AVL⁺11, LKMT11] apply some of these concepts.

In order to be able to process large amounts of data, a robust system able to deal with hundreds of thousands of RDF triples is necessary. We have seen in Chapter 6 that current systems, when they have to deal with large amounts of data, need more time for returning results for a SPARQL query than a DQP based architecture system. We believe that a DQP architecture with distributed nodes for

¹<http://www.w3.org/TR/void/>

7. CONCLUSIONS AND FUTURE WORK

processing the federation instead of a single working node that concentrates all the processing is the path to be followed.

Finally, Linked Data aims to produce a huge network of interconnected data sets. Billions of triples are already available and interconnected, thus to query them is a significant problem due to the large amount of results that may be generated by a single query. The ability to deal with such large amounts of results is key to providing a useful answer to a user. This capability should focus on two concepts, the first is the ability to transfer and process large amounts of data across networks in reasonable amounts of time and secondly, the ability to select the most appropriate results for the user. For a common user, to process the large amounts of results that may come from a single query may be a cumbersome task. Thus, selecting the most appropriate results is a required feature and another line of interesting work if such tools are to become common place and useful.

Appendix A

Appendix A

A.1 Proofs

A.1.1 Proof of Theorem 1

The satisfiability problem for relational algebra is the problem of verifying, given a relational expression φ , whether there exists a (finite) database instance I such that the set of answers of φ over I is not empty. Given that this problem is undecidable [S. 95], it is possible to prove from the results in [AG08] the following result about the complexity of the satisfiability problem for SPARQL. A graph pattern P is said to be satisfiable if there exists an RDF graph G such that $\llbracket P \rrbracket_G^{DS} \neq \emptyset$.

Claim 2. *The problem of verifying, given a graph pattern P , whether P is satisfiable is undecidable.*

Next we show that the complement of the previous problem can be reduced to the problem of verifying, given a graph pattern P and a variable $?X \in \text{var}(P)$, whether $?X$ is bound in P , from which we conclude that the theorem holds.

Let P be a graph pattern and $?X, ?Y, ?Z$ be variables that are not mentioned in P . Then define a graph pattern Q as:

$$Q = ((?X, ?Y, ?Z) \text{ UNION } P).$$

It is easy to see that variable $?X$ is bound in Q if and only if pattern P is not satisfiable, which was to be shown.

A.1.2 Proof of Proposition 1

Let P be a SPARQL query and $?X \in \text{var}(P)$. Next we show that if $?X \in \text{SB}(P)$, then $?X$ is bound in P .

A. APPENDIX A

The proof is by induction on the structure of P . If P is a triple pattern the proposition trivially holds. Now assume that the proposition holds for patterns P_1 and P_2 and consider the following cases:

- Assume that $P = (P_1 \text{ AND } P_2)$ and $?X \in \text{SB}(P)$. Then we have that $\text{SB}(P) = \text{SB}(P_1) \cup \text{SB}(P_2)$ and, therefore, $?X \in \text{SB}(P_1)$ or $?X \in \text{SB}(P_2)$. Without loss of generality assume that $?X \in \text{SB}(P_1)$. Now, let G be an RDF graph and μ be a mapping such that $\mu \in \llbracket P \rrbracket_G^{DS}$. In order to prove that $?X$ is bound in P , we have to demonstrate that $?X \in \text{dom}(\mu)$ and $\mu(?X) \in (\text{dom}(G) \cup \text{dom}(P))$. Given that $\mu \in \llbracket P \rrbracket_G^{DS}$, we have that $\mu = \mu_1 \cup \mu_2$, where $\mu_1 \in \llbracket P_1 \rrbracket_G^{DS}$ and $\mu_2 \in \llbracket P_2 \rrbracket_G^{DS}$. By induction hypothesis, we have that $?X$ is bound in P_1 since $?X \in \text{SB}(P_1)$. Hence, given that $\mu_1 \in \llbracket P_1 \rrbracket_G^{DS}$, we conclude that $?X \in \text{dom}(\mu_1)$ and $\mu_1(?X) \in (\text{dom}(G) \cup \text{dom}(P_1))$. Thus, given that $\mu(?X) = \mu_1(?X)$, $\text{dom}(\mu_1) \subseteq \text{dom}(\mu)$ and $\text{dom}(P_1) \subseteq \text{dom}(P)$, we conclude that $?X \in \text{dom}(\mu)$ and $\mu(?X) \in (\text{dom}(G) \cup \text{dom}(P))$, which was to be shown.
- Assume that $P = (P_1 \text{ UNION } P_2)$ and $?X \in \text{SB}(P)$. Then we have that $\text{SB}(P) = \text{SB}(P_1) \cap \text{SB}(P_2)$ and, therefore, $?X \in \text{SB}(P_1)$ and $?X \in \text{SB}(P_2)$. Now, let G be an RDF graph and μ be a mapping such that $\mu \in \llbracket P \rrbracket_G^{DS}$. In order to prove that $?X$ is bound in P , we have to demonstrate that $?X \in \text{dom}(\mu)$ and $\mu(?X) \in (\text{dom}(G) \cup \text{dom}(P))$. Given that $\mu \in \llbracket P \rrbracket_G^{DS}$, we have that $\mu \in \llbracket P_1 \rrbracket_G^{DS}$ or $\mu \in \llbracket P_2 \rrbracket_G^{DS}$. Assume without loss of generality that $\mu \in \llbracket P_1 \rrbracket_G^{DS}$. By induction hypothesis, we have that $?X$ is bound in P_1 since $?X \in \text{SB}(P_1)$. Hence, given that $\mu \in \llbracket P_1 \rrbracket_G^{DS}$, we conclude that $?X \in \text{dom}(\mu)$ and $\mu(?X) \in (\text{dom}(G) \cup \text{dom}(P_1))$. Thus, given that $\text{dom}(P_1) \subseteq \text{dom}(P)$, we conclude that $?X \in \text{dom}(\mu)$ and $\mu(?X) \in (\text{dom}(G) \cup \text{dom}(P))$, which was to be shown.
- Assume that $P = (P_1 \text{ OPT } P_2)$ and $?X \in \text{SB}(P)$. Then we have that $\text{SB}(P) = \text{SB}(P_1)$ and, therefore, $?X \in \text{SB}(P_1)$. Now, let G be an RDF graph and μ be a mapping such that $\mu \in \llbracket P \rrbracket_G^{DS}$. In order to prove that $?X$ is bound in P , we have to demonstrate that $?X \in \text{dom}(\mu)$ and $\mu(?X) \in (\text{dom}(G) \cup \text{dom}(P))$. Given that $\mu \in \llbracket P \rrbracket_G^{DS}$, we have that either $\mu = \mu_1 \cup \mu_2$, where $\mu_1 \in \llbracket P_1 \rrbracket_G^{DS}$ and $\mu_2 \in \llbracket P_2 \rrbracket_G^{DS}$, or $\mu \in \llbracket P_1 \rrbracket_G^{DS}$ and μ is not compatible with any mapping in $\llbracket P_2 \rrbracket_G^{DS}$.
 - In the first case, given that $?X$ is bound in P_1 (since $?X \in \text{SB}(P_1)$) and $\mu_1 \in \llbracket P_1 \rrbracket_G^{DS}$, we have that $?X \in \text{dom}(\mu_1)$ and $\mu_1(?X) \in (\text{dom}(G) \cup \text{dom}(P_1))$. Thus, given that $\mu(?X) = \mu_1(?X)$, $\text{dom}(\mu_1) \subseteq \text{dom}(\mu)$ and $\text{dom}(P_1) \subseteq \text{dom}(P)$, we conclude that $?X \in \text{dom}(\mu)$ and $\mu(?X) \in (\text{dom}(G) \cup \text{dom}(P))$, which was to be shown.

- In the second case, given that $?X$ is bound in P_1 (since $?X \in \text{SB}(P_1)$) and $\mu \in \llbracket P_1 \rrbracket_G^{DS}$, we have that $?X \in \text{dom}(\mu)$ and $\mu(?X) \in (\text{dom}(G) \cup \text{dom}(P_1))$. Thus, given that $\text{dom}(P_1) \subseteq \text{dom}(P)$, we conclude that $?X \in \text{dom}(\mu)$ and $\mu(?X) \in (\text{dom}(G) \cup \text{dom}(P))$, which was to be shown.
- Assume that $P = (P_1 \text{ FILTER } R)$, where R is a built-in condition, and that $?X \in \text{SB}(P)$. Then we have that $\text{SB}(P) = \text{SB}(P_1)$ and, therefore, $?X \in \text{SB}(P_1)$. Now, let G be an RDF graph and μ be a mapping such that $\mu \in \llbracket P \rrbracket_G^{DS}$. In order to prove that $?X$ is bound in P , we have to demonstrate that $?X \in \text{dom}(\mu)$ and $\mu(?X) \in (\text{dom}(G) \cup \text{dom}(P))$. Given that $\mu \in \llbracket P \rrbracket_G^{DS}$, we have that $\mu \in \llbracket P_1 \rrbracket_G^{DS}$ and $\mu \models R$. By induction hypothesis, we have that $?X$ is bound in P_1 since $?X \in \text{SB}(P_1)$. Hence, given that $\mu \in \llbracket P_1 \rrbracket_G^{DS}$, we conclude that $?X \in \text{dom}(\mu)$ and $\mu(?X) \in (\text{dom}(G) \cup \text{dom}(P_1))$. Thus, given that $\text{dom}(P_1) \subseteq \text{dom}(P)$, we conclude that $?X \in \text{dom}(\mu)$ and $\mu(?X) \in (\text{dom}(G) \cup \text{dom}(P))$, which was to be shown.
- Assume that $P = (\text{SERVICE } c \ P_1)$, where $c \in I$, or $P = (\text{SERVICE } ?X \ P_1)$, where $?X \in V$. Then given $\text{SB}(P) = \emptyset$, we conclude that the property trivially holds.
- Assume that $P = (P_1 \text{ BINDINGS } L \ \{A_1, \dots, A_n\})$ and $?X \in \text{SB}(P)$. Then, given that

$$\begin{aligned} \text{SB}(P) = \text{SB}(P_1) \cup \{?Y \mid ?Y \text{ is a member of the list } L \text{ and} \\ \text{for every } i \in \{1, \dots, n\}, \text{ it holds that } ?Y \in \text{dom}(\mu_{L, A_i})\}, \end{aligned}$$

we conclude that either $?X \in \text{SB}(P_1)$ or $?X$ is a member of the list L and for every $i \in \{1, \dots, n\}$, it holds that $?X \in \text{dom}(\mu_{L, A_i})$. Now, let G be an RDF graph and μ be a mapping such that $\mu \in \llbracket P \rrbracket_G^{DS}$. In order to prove that $?X$ is bound in P , we have to demonstrate that $?X \in \text{dom}(\mu)$ and $\mu(?X) \in (\text{dom}(G) \cup \text{dom}(P))$. Given that $\mu \in \llbracket P \rrbracket_G^{DS}$, we have that there exist $\mu_1 \in \llbracket P_1 \rrbracket_G^{DS}$ and $k \in \{1, \dots, n\}$ such that $\mu = \mu_1 \cup \mu_{L, A_k}$. Next we consider two cases to prove that $?X \in \text{dom}(\mu)$ and $\mu(?X) \in (\text{dom}(G) \cup \text{dom}(P))$.

- First, assume that $?X \in \text{SB}(P_1)$. Then by induction hypothesis we have that $?X$ is bound in P_1 (since $?X \in \text{SB}(P_1)$). Thus, given that $\mu_1 \in \llbracket P_1 \rrbracket_G^{DS}$, we conclude that $?X \in \text{dom}(\mu_1)$ and $\mu_1(?X) \in (\text{dom}(G) \cup \text{dom}(P_1))$. Therefore, given that $\text{dom}(\mu_1) \subseteq \text{dom}(\mu)$, $\mu(?X) = \mu_1(?X)$ and $\text{dom}(P_1) \subseteq \text{dom}(P)$, we conclude that $?X \in \text{dom}(\mu)$ and $\mu(?X) \in (\text{dom}(G) \cup \text{dom}(P))$, which was to be shown.
- Second, assume that $?X$ is a member of the list L and for every $i \in \{1, \dots, n\}$, it holds that $?X \in \text{dom}(\mu_{L, A_i})$. Then we have that $?X \in \text{dom}(\mu_{L, A_k})$, which implies that $\mu(?X) \in$

$\text{dom}(P)$ since if $a \in (I \cup L)$ is mentioned in A_i , then a is in $\text{dom}(P)$. Thus, given that $\text{dom}(\mu_{L,A_k}) \subseteq \text{dom}(\mu)$ and $\mu(?X) = \mu_{L,A_k}(?X)$, we conclude that $?X \in \text{dom}(\mu)$ and $\mu(?X) \in (\text{dom}(G) \cup \text{dom}(P))$, which was to be shown.

- Assume that $P = (\text{SELECT } W \ P_1)$ and $?X \in \text{SB}(P)$. Then we have that $\text{SB}(P) = (W \cap \text{SB}(P_1))$ and, therefore, $?X \in W$ and $?X \in \text{SB}(P_1)$. Now, let G be an RDF graph and μ be a mapping such that $\mu \in \llbracket P \rrbracket_G^{DS}$. In order to prove that $?X$ is bound in P , we have to demonstrate that $?X \in \text{dom}(\mu)$ and $\mu(?X) \in (\text{dom}(G) \cup \text{dom}(P))$. Given that $\mu \in \llbracket P \rrbracket_G^{DS}$, we have that there exists $\mu^* \in \llbracket P_1 \rrbracket_G^{DS}$ such that $\mu|_W = \mu^*$. By induction hypothesis, we have that $?X$ is bound in P_1 since $?X \in \text{SB}(P_1)$. Hence, given that $\mu^* \in \llbracket P_1 \rrbracket_G^{DS}$, we conclude that $?X \in \text{dom}(\mu^*)$ and $\mu^*(?X) \in (\text{dom}(G) \cup \text{dom}(P_1))$. Thus, given that $?X \in W$, $\text{dom}(\mu) = (\text{dom}(\mu^*) \cap W)$, $\mu(?X) = \mu^*(?X)$ and $\text{dom}(P_1) \subseteq \text{dom}(P)$, we conclude that $?X \in \text{dom}(\mu)$ and $\mu(?X) \in (\text{dom}(G) \cup \text{dom}(P))$, which was to be shown.

A.1.3 Proof of Theorem 2

As in the proof of Theorem 1, we use the undecidability of the satisfiability problem for SPARQL to show that the theorem holds. Let P be a SPARQL graph pattern and $?X, ?Y, ?Z, ?U, ?V, ?W$ be variables that are not mentioned in P , and assume that P does not mention the operator SERVICE (recall that the satisfiability problem is already undecidable for the fragment of SPARQL consisting of the operators AND, UNION, OPT and FILTER). Then define a SPARQL query Q as:

$$Q = \left(\left((?X, ?Y, ?Z) \text{ UNION } P \right) \text{ AND } \left(\text{SERVICE } ?X (?U, ?V, ?W) \right) \right).$$

Next we show that Q is service-bound if and only if P is not satisfiable.

(\Leftarrow) If P is not satisfiable, then Q is equivalent to the pattern:

$$Q' = ((?X, ?Y, ?Z) \text{ AND } (\text{SERVICE } ?X (?U, ?V, ?W))),$$

which is service-bound since variable $?X$ is bound in Q' .

(\Rightarrow) Assume that P is satisfiable. Then given that variable $?X$ is not mentioned in P , we have that $?X$ is not bound in the graph pattern $((?X, ?Y, ?Z) \text{ UNION } P)$. Thus, given that $?X$ is neither bounded in $(\text{SERVICE } ?X (?U, ?V, ?W))$, we deduce that query Q is not service-bound since $?X$ is not a bound variable in Q .

Therefore, we have shown that the complement of the satisfiability problem for SPARQL can be reduced to the problem of verifying, given a SPARQL query P , whether P is service-bound. From this we conclude that the theorem holds.

A.1.4 Proof of Proposition 2

This proposition is a corollary of Proposition 1.

Proof of Claim 1

The proof is by induction on the structure of pattern Q . If Q is a triple pattern, then the property trivially holds. For the inductive step, first suppose that $Q = (Q_1 \text{ AND } Q_2)$. Since $?X \in \text{var}(Q')$ for every $Q' \sqsubseteq Q$, we have that $?X \in \text{var}(Q'_1)$ for every $Q'_1 \sqsubseteq Q_1$, or $?X \in \text{var}(Q'_2)$ for every $Q'_2 \sqsubseteq Q_2$. Assume, without loss of generality, that $?X \in \text{var}(Q'_1)$ for every $Q'_1 \sqsubseteq Q_1$. Then by induction hypothesis, we have that $?X \in \text{dom}(\mu_1)$ for every $\mu_1 \in \llbracket Q_1 \rrbracket_G^{DS}$, from which we conclude that $?X \in \text{dom}(\mu)$ for every $\mu \in \llbracket (Q_1 \text{ AND } Q_2) \rrbracket_G^{DS}$. Second, suppose that $Q = (Q_1 \text{ OPT } Q_2)$. Since $?X \in \text{var}(Q')$ for every $Q' \sqsubseteq Q$, and since $Q_1 \sqsubseteq Q$, we have that $?X \in \text{var}(Q'_1)$ for every $Q'_1 \sqsubseteq Q_1$. By induction hypothesis, $?X \in \text{dom}(\mu)$ for all $\mu \in \llbracket Q_1 \rrbracket_G^{DS}$ and, hence, by the definition of OPT, we conclude that $?X \in \text{dom}(\mu)$ for every $\mu \in \llbracket (Q_1 \text{ OPT } Q_2) \rrbracket_G^{DS}$. Third, suppose that $Q = (Q_1 \text{ FILTER } R)$. Since Q is safe, if $?X \in \text{var}(Q)$ then $?X \in \text{var}(Q_1)$, and, thus, we have that $?X \in \text{var}(Q'_1)$ for every $Q'_1 \sqsubseteq Q_1$. By induction hypothesis, we have that $?X \in \text{dom}(\mu)$ for every $\mu \in \llbracket Q_1 \rrbracket_G^{DS}$ and thus, given that $\llbracket (Q_1 \text{ FILTER } R) \rrbracket_G^{DS} \subseteq \llbracket Q_1 \rrbracket_G^{DS}$, we conclude that $?X \in \text{dom}(\mu)$ for every $\mu \in \llbracket (Q_1 \text{ FILTER } R) \rrbracket_G^{DS}$. Fourth, suppose that $Q = (\text{SERVICE } a \text{ } O)$, where O is a graph pattern. If $?X \in \text{var}(Q)$, then $?X \in \text{var}(O)$. Thus, if we assume that $?X$ is a variable such that $?X \in \text{var}(Q')$, for every $Q' \sqsubseteq Q$, then we have that $?X \in \text{var}(O')$ for every $O' \sqsubseteq O$. Hence, we have by induction hypothesis that $?X \in \text{dom}(\mu)$ for every $\mu \in \llbracket O \rrbracket_{ep(a)}$, and, therefore, $?X \in \text{dom}(\mu)$ for every $\mu \in \llbracket O \rrbracket_{ep(a)}$ (since $\llbracket Q \rrbracket_G^{DS} = \llbracket O \rrbracket_{ep(a)}$).

A.2 Evaluation 1 Queries

A.2.1 Bio2RDF Queries

The following are the queries used in the Bio2RDF evaluation:

```
Q1 SELECT ?pubmed ?gene1 ?mesh ?descriptor WHERE
{
  SERVICE <http://127.0.0.1:2020/sparql-pubmed> {
    ?gene1 <http://bio2rdf.org/geneid_resource:pubmed_xref> ?pubmed .}.
  SERVICE <http://pubmed.bio2rdf.org/sparql> {
    ?pubmed <http://bio2rdf.org/pubmed_resource:meshref> ?mesh .
    ?mesh <http://bio2rdf.org/pubmed_resource:descriptor> ?descriptor .}.
}
```

A. APPENDIX A

Q2 SELECT ?pubmed ?gene1 ?mesh ?descriptor WHERE
{
 SERVICE <http://127.0.0.1:2020/sparql-pubmed> {
 ?gene1 <http://bio2rdf.org/geneid_resource:pubmed_xref> ?pubmed .}.
 OPTIONAL {
 SERVICE <http://pubmed.bio2rdf.org/sparql> {
 ?pubmed <http://bio2rdf.org/pubmed_resource:meshref> ?mesh .
 ?mesh <http://bio2rdf.org/pubmed_resource:descriptor> ?descriptor .}}.
}

Q3 SELECT ?pubmed ?gene1 ?mesh ?descriptor ?meshReference WHERE
{
 SERVICE <http://127.0.0.1:2020/sparql-pubmed> {
 ?gene1 <http://bio2rdf.org/geneid_resource:pubmed_xref> ?pubmed .}.
 SERVICE <http://pubmed.bio2rdf.org/sparql> {
 ?pubmed <http://bio2rdf.org/pubmed_resource:meshref> ?mesh .
 ?mesh <http://bio2rdf.org/pubmed_resource:descriptor> ?descriptor .}.
 SERVICE <http://127.0.0.1:2021/sparql-mesh> {
 ?meshReference <http://www.w3.org/2002/07/owl#sameAs> ?descriptor .}.
}

Q4 SELECT ?pubmed ?gene1 ?mesh ?descriptor ?meshReference WHERE
{
 SERVICE <http://127.0.0.1:2020/sparql-pubmed> {
 ?gene1 <http://bio2rdf.org/geneid_resource:pubmed_xref> ?pubmed .}.
 SERVICE <http://pubmed.bio2rdf.org/sparql> {
 ?pubmed <http://bio2rdf.org/pubmed_resource:meshref> ?mesh .
 ?mesh <http://bio2rdf.org/pubmed_resource:descriptor> ?descriptor .}.
 OPTIONAL {
 SERVICE <http://127.0.0.1:2021/sparql-mesh> {
 ?meshReference <http://www.w3.org/2002/07/owl#sameAs> ?descriptor .} }.
}

Q5 SELECT ?pubmed ?gene1 ?mesh ?descriptor ?meshReference WHERE
{
 SERVICE <http://quebec.hhp.id.bio2rdf.org/sparql> {
 ?interaction <http://ontology.bio2rdf.org/hhp.id:elementGene2> ?gene1 .}
 SERVICE <http://127.0.0.1:2020/sparql-pubmed> {
 ?gene1 <http://bio2rdf.org/geneid_resource:pubmed_xref> ?pubmed .}.
 SERVICE <http://pubmed.bio2rdf.org/sparql> {
 ?pubmed <http://bio2rdf.org/pubmed_resource:meshref> ?mesh .
 ?mesh <http://bio2rdf.org/pubmed_resource:descriptor> ?descriptor .}.
 SERVICE <http://127.0.0.1:2021/sparql-mesh> {
 ?meshReference <http://www.w3.org/2002/07/owl#sameAs> ?descriptor .}.
}

Q6 SELECT ?interaction ?pubmed ?gene1 ?mesh ?descriptor ?meshReference WHERE
{
 SERVICE <http://quebec.hhp.id.bio2rdf.org/sparql> {
 ?interaction <http://ontology.bio2rdf.org/hhp.id:elementGene2> ?gene1 .}
 SERVICE <http://127.0.0.1:2020/sparql-pubmed> {
 ?gene1 <http://bio2rdf.org/geneid_resource:pubmed_xref> ?pubmed .}.
 SERVICE <http://pubmed.bio2rdf.org/sparql> {
 ?pubmed <http://bio2rdf.org/pubmed_resource:meshref> ?mesh .
 ?mesh <http://bio2rdf.org/pubmed_resource:descriptor> ?descriptor .}.
 OPTIONAL {
 SERVICE <http://127.0.0.1:2021/sparql-mesh> {
 ?meshReference <http://www.w3.org/2002/07/owl#sameAs> ?descriptor .}}.
}

Q7

```

SELECT ?interaction ?pubmed ?gene1 ?mesh ?descriptor ?meshReference WHERE
{
  SERVICE <http://quebec.hhp.id.bio2rdf.org/sparql> {
    ?interaction <http://ontology.bio2rdf.org/hhp.id:elementGene2> ?gene1 .
  }
  SERVICE <http://127.0.0.1:2020/sparql-pubmed> {
    ?gene1 <http://bio2rdf.org/geneid_resource:pubmed_xref> ?pubmed .
  }
  OPTIONAL {
    SERVICE <http://pubmed.bio2rdf.org/sparql> {
      ?pubmed <http://bio2rdf.org/pubmed_resource:meshref> ?mesh .
      ?mesh <http://bio2rdf.org/pubmed_resource:descriptor> ?descriptor .
    }
    SERVICE <http://127.0.0.1:2021/sparql-mesh> {
      ?meshReference <http://www.w3.org/2002/07/owl#sameAs> ?descriptor .
    }
  }
}

```

A.3 Evaluation 2 Queries

A.3.1 Cross Domain Queries

Q1

```

PREFIX owl: <http://www.w3.org/2002/07/owl#>
SELECT ?predicate ?object WHERE {
  {
    SERVICE<http://dbpedia.org/sparql> {
      <http://dbpedia.org/resource/Barack_Obama> ?predicate ?object
    }
  }
  UNION
  {
    SERVICE <http://api.talis.com/stores/nytimes/services/sparql> {
      ?subject owl:sameAs <http://dbpedia.org/resource/Barack_Obama> .
      ?subject ?predicate ?object
    }
  }
}

```

Q2

```

PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX dbont: <http://dbpedia.org/ontology/>
SELECT ?party ?page ?x WHERE {
  SERVICE<http://dbpedia.org/sparql> {
    <http://dbpedia.org/resource/Barack_Obama> dbont:party ?party .
  }
  SERVICE<http://api.talis.com/stores/nytimes/services/sparql> {
    ?x <http://data.nytimes.com/elements/topicPage> ?page .
    ?x owl:sameAs <http://dbpedia.org/resource/Barack_Obama> .
  }
}

```

Q3

```

PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dbont: <http://dbpedia.org/ontology/>
SELECT ?president ?party ?page WHERE {
  SERVICE<http://dbpedia.org/sparql> {
    ?president rdf:type dbont:President .
    ?president dbont:nationality <http://dbpedia.org/resource/United_States> .
    ?president dbont:party ?party .
  }
  SERVICE<http://api.talis.com/stores/nytimes/services/sparql> {
    ?x <http://data.nytimes.com/elements/topicPage> ?page .
    ?x owl:sameAs ?president .
  }
}

```

A. APPENDIX A

```
}  
}
```

Q4

```
PREFIX owl: <http://www.w3.org/2002/07/owl#>  
SELECT ?actor ?news WHERE {  
  SERVICE<http://data.linkedmdb.org/sparql> {  
    ?film dct:terms:title 'Tarzan' .  
    ?film <http://data.linkedmdb.org/resource/movie/actor> ?actor .  
    ?actor owl:sameAs ?x .  
  }  
  SERVICE<http://api.talis.com/stores/nytimes/services/sparql> {  
    ?y <http://www.w3.org/2002/07/owl#sameAs> ?x .  
    ?y <http://data.nytimes.com/elements/topicPage> ?news  
  }  
}
```

Q4b

```
PREFIX dct: <http://purl.org/dc/terms/>  
PREFIX imdb: <http://data.linkedmdb.org/resource/movie/>  
PREFIX owl: <http://www.w3.org/2002/07/owl#>  
SELECT ?title ?actor ?news ?director ?film WHERE {  
  SERVICE<http://data.linkedmdb.org/sparql> {  
    ?film dct:terms:title ?title .  
    ?film imdb:actor ?actor .  
    ?film imdb:production_company <http://data.linkedmdb.org/resource/production_company/15>  
    ?actor owl:sameAs ?x .  
  }  
  OPTIONAL {  
    SERVICE<http://api.talis.com/stores/nytimes/services/sparql> {  
      ?y owl:sameAs ?x .  
      ?y <http://data.nytimes.com/elements/topicPage> ?news  
    }  
  }  
  FILTER (?title="Tarzan")  
}
```

Q5

```
PREFIX owl: <http://www.w3.org/2002/07/owl#>  
PREFIX imdb: <http://data.linkedmdb.org/resource/movie>  
PREFIX dbont: <http://dbpedia.org/ontology/>  
SELECT ?film ?director ?genre ?x WHERE {  
  SERVICE<http://dbpedia.org/sparql> {  
    ?film dbont:director ?director .  
    ?director dbont:nationality <http://dbpedia.org/resource/Italy> .  
  }  
  SERVICE<http://data.linkedmdb.org/sparql> {  
    ?x owl:sameAs ?film .  
    ?x imdb:genre ?genre .  
  }  
}
```

Q6

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>  
PREFIX geo: <http://www.geonames.org/ontology#>  
SELECT ?name ?location ?news WHERE {  
  SERVICE<http://linkedgeodata.org/sparql> {  
    ?artist foaf:name ?name .  
    ?artist foaf:based_near ?location .  
    ?location geo:parentFeature ?germany .  
    ?germany geo:name 'Federal Republic of Germany'  
  }  
}
```


Q7

```

SELECT ?location ?news WHERE {
  SERVICE<http://linkedgeodata.org/sparql> {
    ?location <http://www.geonames.org/ontology#parentFeature> ?parent .
    ?parent <http://www.geonames.org/ontology#name> 'California' .
  }
  SERVICE<http://api.talis.com/stores/nytimes/services/sparql> {
    ?y <http://www.w3.org/2002/07/owl#sameAs> ?location .
    ?y <http://data.nytimes.com/elements/topicPage> ?news
  }
}

```

Q8

```

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdf:      <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dbprop: <http://dbpedia.org/property/>
PREFIX viajero: <http://webenemasuno.linkeddata.es/ontology/OPMO/>
SELECT ?Book ?Country ?Area
WHERE {
  SERVICE <http://dbpedia.org/sparql> {
    ?Country rdf:type <http://dbpedia.org/ontology/Country> .
    ?Country dbprop:areaKm ?Area
  }
  OPTIONAL {
    SERVICE <http://webenemasuno.linkeddata.es/sparql> {
      ?Book viajero:refersTo ?Country .
    }
  }
  FILTER(?Area < "20000"^^xsd:integer)
}

```

Q9

```

PREFIX rdf:      <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dbprop: <http://dbpedia.org/property/>
PREFIX dbont: <http://dbpedia.org/ontology/>
PREFIX viajero: <http://webenemasuno.linkeddata.es/ontology/OPMO/>
PREFIX factbook: <http://www4.wiwiss.fu-berlin.de/factbook/ns#>

SELECT ?Book ?Country ?Area ?climate
WHERE {
  SERVICE <http://dbpedia.org/sparql> {
    ?Country rdf:type dbont:Country .
    ?Country dbprop:areaKm ?Area .
    ?Country rdfs:label ?countryLabel
  }
  OPTIONAL {
    SERVICE <http://webenemasuno.linkeddata.es/sparql> {
      ?Book viajero:refersTo ?Country .
    }
  }
  SERVICE <http://www4.wiwiss.fu-berlin.de/factbook/sparql> {
    ?CountryCIA rdfs:label ?countryLabel .
    ?CountryCIA rdf:type factbook:Country .
    ?CountryCIA factbook:climate ?climate
  }
}

```

A. APPENDIX A

A.3.2 Life Sciences Queries

Q1

```
PREFIX drugbank: <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/>
SELECT ?drug ?melt WHERE {
  {
    SERVICE <http://www4.wiwiss.fu-berlin.de/drugbank/sparql> {
      ?drug drugbank:meltingPoint ?melt .
    }
  }
  UNION
  {
    SERVICE <http://dbpedia.org/sparql> {
      ?drug <http://dbpedia.org/ontology/Drug/meltingPoint> ?melt .
    }
  }
}
```

Q2

```
PREFIX drugs: <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugs/>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
SELECT ?predicate ?object WHERE {
  {
    SERVICE <http://www4.wiwiss.fu-berlin.de/drugbank/sparql> {
      drugs:DB00201 ?predicate ?object .
    }
  }
  UNION
  {
    SERVICE <http://www4.wiwiss.fu-berlin.de/drugbank/sparql> {
      drugs:DB00201 owl:sameAs ?caff .
      ?caff ?predicate ?object .
    }
  }
}
```

Q3

```
PREFIX drugbank: <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/>
PREFIX dbont: <http://dbpedia.org/ontology/>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?Drug ?IntDrug ?IntEffect
WHERE {
  SERVICE <http://dbpedia.org/sparql> {
    ?Drug rdf:type dbont:Drug .
    ?Drug owl:sameAs ?y .
  }
  SERVICE <http://www4.wiwiss.fu-berlin.de/drugbank/sparql> {
    ?Int drugbank:interactionDrug1 ?y .
    ?Int drugbank:interactionDrug2 ?IntDrug .
    ?Int drugbank:text ?IntEffect .
  }
}
```

Q4

```
PREFIX cat: <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugcategory/>
PREFIX drugbank: <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/>
PREFIX kegg: <http://bio2rdf.org/ns/kegg#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?drugDesc ?cpd ?equation WHERE {
  SERVICE <http://www4.wiwiss.fu-berlin.de/drugbank/sparql> {
    ?drug drugbank:drugCategory cat:cathartics .
    ?drug drugbank:keggCompoundId ?cpd .
    ?drug drugbank:description ?drugDesc .
  }
  SERVICE <http://kegg.bio2rdf.org/sparql> {
    ?enzyme kegg:xSubstrate ?cpd .
  }
}
```

```

        ?enzyme rdf:type kegg:nzyme .
        ?reaction kegg:xEnzyme ?enzyme .
        ?reaction kegg:equation ?equation .
    }
}

```

Q5

```

PREFIX bio: <http://bio2rdf.org/ns/bio2rdf#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX drugbank: <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?drug ?keggUrl ?chebiImage WHERE {
    SERVICE <http://www4.wiwiss.fu-berlin.de/drugbank/sparql> {
        ?drug rdf:type drugbank:drugs .
        ?drug drugbank:keggCompoundId ?keggDrug .
    }
    SERVICE <http://kegg.bio2rdf.org/sparql> {
        ?keggDrug bio:url ?keggUrl .
        ?drug drugbank:genericName ?drugBankName .
        ?chebiDrug dc:title ?drugBankName .
        ?chebiDrug bio:image ?chebiImage .
    }
}

```

Q6

```

PREFIX cat: <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugcategory/>
PREFIX drugbank: <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/>
PREFIX kegg: <http://bio2rdf.org/ns/kegg#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX bio: <http://bio2rdf.org/ns/bio2rdf#>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT DISTINCT ?drug ?title WHERE {
    SERVICE <http://www4.wiwiss.fu-berlin.de/drugbank/sparql> {
        ?drug drugbank:drugCategory cat:micronutrient .
        ?drug drugbank:casRegistryNumber ?id .
    }
    SERVICE <http://kegg.bio2rdf.org/sparql> {
        ?keggDrug rdf:type kegg:Drug .
        ?keggDrug bio:xRef ?id .
        ?keggDrug dc:title ?title .
    }
}

```

Q7

```

PREFIX bio: <http://bio2rdf.org/ns/bio2rdf#>
PREFIX drugbank: <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/>
SELECT ?drug ?transform ?mass WHERE {
    SERVICE <http://www4.wiwiss.fu-berlin.de/drugbank/sparql> {
        ?drug drugbank:affectedOrganism 'Humans and other mammals'.
        ?drug drugbank:casRegistryNumber ?cas .
    }
    SERVICE <http://kegg.bio2rdf.org/sparql> {
        ?keggDrug bio:xRef ?cas .
        ?keggDrug bio:mass ?mass
        FILTER ( ?mass > '5' )
    }
    OPTIONAL {
        SERVICE <http://www4.wiwiss.fu-berlin.de/drugbank/sparql> {
            ?drug drugbank:biotransformation ?transform .
        }
    }
}

```

A. APPENDIX A

A.3.3 SP2Bench Queries

The following are the queries used in the evaluation:

Q1

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX bench: <http://localhost/vocabulary/bench/>
PREFIX dcterms: <http://purl.org/dc/terms/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?yr
WHERE {
  SERVICE <http://localhost:3030/Journal/sparql> {
    ?journal rdf:type bench:Journal .
    ?journal dc:title "Journal 1 (1940)"^^<http://www.w3.org/2001/XMLSchema#string> .
    ?journal dcterms:issued ?yr }
}
```

Q2

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX bench: <http://localhost/vocabulary/bench/>
PREFIX dcterms: <http://purl.org/dc/terms/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX swrc: <http://swrc.ontoware.org/ontology#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?inproc ?author ?booktitle ?title
       ?proc ?ee ?page ?url ?yr ?abstract
WHERE {
  SERVICE <http://localhost:3030/InProceedings/sparql> {
    ?inproc rdf:type bench:Inproceedings .
    ?inproc dc:creator ?author .
    ?inproc bench:booktitle ?booktitle .
    ?inproc dc:title ?title .
    ?inproc dcterms:partOf ?proc .
    ?inproc rdfs:seeAlso ?ee .
    ?inproc swrc:pages ?page .
    ?inproc foaf:homepage ?url .
    ?inproc dcterms:issued ?yr
    OPTIONAL {
      ?inproc bench:abstract ?abstract
    }
  }
}
```

Q3

```
PREFIX bench: <http://localhost/vocabulary/bench/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?article
WHERE {
  SERVICE <http://localhost:3030/InProceedings/sparql> {
    ?article rdf:type bench:Article .
    ?article ?property ?value
    FILTER (?property=<http://swrc.ontoware.org/ontology#pages>) }
}
```

Q4

```
PREFIX bench: <http://localhost/vocabulary/bench/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?article
WHERE {
  SERVICE <http://localhost:3030/Journal/sparql> {
```

```

?article rdf:type bench:Article .
?article ?property ?value
FILTER (?property=<http://swrc.ontoware.org/ontology#month>)
}
}

```

Q5

```

PREFIX bench: <http://localhost/vocabulary/bench/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?article
WHERE {
  SERVICE <http://localhost:3030/InProceedings/sparql> {
    ?article rdf:type bench:Article .
    ?article ?property ?value
    FILTER (?property=<http://swrc.ontoware.org/ontology#isbn>)
  }
}

```

Q6

```

PREFIX bench: <http://localhost/vocabulary/bench/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX swrc: <http://swrc.ontoware.org/ontology#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT DISTINCT ?name1 ?name2 WHERE {
  SERVICE <http://localhost:3030/Journal/sparql> {
    ?article1 rdf:type bench:Article .
    ?article2 rdf:type bench:Article .
    ?article1 dc:creator ?author1 .
    ?article1 swrc:journal ?journal .
    ?article2 dc:creator ?author2 .
    ?article2 swrc:journal ?journal
  }
  SERVICE <http://localhost:3030/People/sparql> {
    ?author1 foaf:name ?name .
    ?author2 foaf:name ?name .
  }
}

```

Q7

```

PREFIX bench: <http://localhost/vocabulary/bench/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT DISTINCT ?person ?name WHERE {
  SERVICE <http://localhost:3030/Journal/sparql> {
    ?article rdf:type bench:Article .
    ?article dc:creator ?person .
  }
  SERVICE <http://localhost:3030/InProceedings/sparql> {
    ?inproc rdf:type bench:Inproceedings .
    ?inproc dc:creator ?person .
  }
  SERVICE <http://localhost:3030/People/sparql> {
    ?person foaf:name ?name .
  }
}

```

Q7b

```

PREFIX bench: <http://localhost/vocabulary/bench/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

```

A. APPENDIX A

```
SELECT DISTINCT ?article ?inproc ?person ?name
WHERE {
  SERVICE <http://localhost:3030/Journal/sparql> {
    ?article rdf:type bench:Article .
    ?article dc:creator ?person .
  }
  OPTIONAL {
    SERVICE <http://localhost:3030/InProceedings/sparql> {
      ?inproc rdf:type bench:Inproceedings .
      ?inproc dc:creator ?person .
    }
  }
  SERVICE <http://localhost:3030/People/sparql> {
    ?person foaf:name ?name .
  }
}
```

Q7c

```
PREFIX bench: <http://localhost/vocabulary/bench/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX swrc: <http://swrc.ontoware.org/ontology#>
SELECT DISTINCT ?article ?inproc ?person ?name ?title ?pages
WHERE {
  SERVICE <http://localhost:3030/Journal/sparql> {
    ?article rdf:type bench:Article .
    ?article dc:creator ?person .
    ?article swrc:pages ?pages
  }
  OPTIONAL {
    SERVICE <http://localhost:3030/People/sparql> {
      ?person foaf:name ?name .
    }
  }
  FILTER (?pages = 200)
}
```

Q8

```
PREFIX bench: <http://localhost/vocabulary/bench/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT DISTINCT ?person ?name ?incol ?inproc
WHERE {
  SERVICE <http://localhost:3030/InCol/sparql> {
    ?incol rdf:type bench:Incollection .
    ?incol dc:creator ?person .
  }
  SERVICE <http://localhost:3030/InProceedings/sparql> {
    ?inproc rdf:type bench:Inproceedings .
    ?inproc dc:creator ?person .
  }
  SERVICE <http://localhost:3030/People/sparql> {
    ?person foaf:name ?name
  }
}
```

Q8b

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX bench: <http://localhost/vocabulary/bench/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
```

```

PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT DISTINCT ?person ?name ?incol ?inproc
WHERE {
  SERVICE <http://localhost:3030/People/sparql> {
    ?person foaf:name ?name
  }
  OPTIONAL {
    SERVICE <http://localhost:3030/InProceedings/sparql> {
      ?inproc rdf:type bench:Inproceedings .
      ?inproc dc:creator ?person .
    }
  }
  SERVICE <http://localhost:3030/InCol/sparql> {
    ?incol rdf:type bench:Incollection .
    ?incol dc:creator ?person .
  }
}

```

Q8c

```

PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX bench: <http://localhost/vocabulary/bench/>
PREFIX dcterms: <http://purl.org/dc/terms/>
PREFIX swrc: <http://swrc.ontoware.org/ontology#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT DISTINCT ?person ?incol ?inproc ?title ?article
WHERE {
  SERVICE <http://localhost:3030/InProceedings/sparql> {
    ?inproc rdf:type <http://localhost/vocabulary/bench/Inproceedings> .
    ?inproc dc:creator ?person .
    ?inproc bench:booktitle ?booktitle .
    ?inproc dc:title ?title .
    ?inproc dcterms:partOf ?proc .
    ?inproc rdfs:seeAlso ?ee .
    ?inproc swrc:pages ?page .
    ?inproc foaf:homepage ?url .
    ?inproc dcterms:issued ?yr
    OPTIONAL {
      ?inproc bench:abstract ?abstract
    }
  }
  OPTIONAL {
    SERVICE <http://localhost:3030/People/sparql> {
      ?person foaf:name ?name
    }
  }
  SERVICE <http://localhost:3030/Journal/sparql> {
    ?article rdf:type <http://localhost/vocabulary/bench/Article> .
    ?article dc:creator ?person .
    ?article ?property ?value .
    FILTER (?property=<http://swrc.ontoware.org/ontology#month>)
  }
  SERVICE <http://localhost:3030/InCol/sparql> {
    ?incol rdf:type <http://localhost/vocabulary/bench/Incollection> .
    ?incol dc:creator ?person .
  }
}

```

A. APPENDIX A

Q8d

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX dcterms: <http://purl.org/dc/terms/>

SELECT DISTINCT ?title ?name
WHERE {
  SERVICE <http://localhost:3030/InProceedings/sparql> {
    ?inproc rdf:type <http://localhost/vocabulary/bench/Inproceedings> .
    ?inproc dc:creator ?person .
    ?inproc dc:title ?title .
    ?inproc dcterms:issued ?yr
  }
  OPTIONAL {
    SERVICE <http://localhost:3030/People/sparql> {
      ?person foaf:name ?name
    }
  }
  FILTER (?yr="1950")
}
```


References

- [AG08] R. Angles and C. Gutierrez. The expressive power of sparql. In *International Semantic Web Conference*, pages 114–129, 2008. [119](#)
- [AH00] Ron Avnur and Joseph M. Hellerstein. Eddies: Continuously adaptive query processing. In Weidong Chen, Jeffrey F. Naughton, and Philip A. Bernstein, editors, *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 261–272. ACM, 2000. [35](#), [36](#), [37](#)
- [AKP⁺06] Mario Antonioletti, Amy Krause, Norman W. Paton, Andrew Eisenberg, Simon Laws, Susan Malaika, Jim Melton, and Dave Pearson. The ws-dai family of specifications for web service data access and integration. *SIGMOD Rec.*, 35:48–55, March 2006. [67](#)
- [AMMH09] Daniel J. Abadi, Adam Marcus, Samuel R. Madden, and Kate Hollenbach. Sw-store: a vertically partitioned dbms for semantic web data management. *The VLDB Journal*, 18:385–406, April 2009. [37](#)
- [Ant07] N.P.C.; Hume A.C.; Jackson M.; Karasavvas K.; Krause A.; Schopf J.M.; Atkinson M.P.; Dobrzelecki B.; Illingworth M.; McDonnell N.; Parsons M.; Theocharopoulos E. Antonioletti, M.; Hong. Ogsa-dai 3.0 - the whats and the whys. In *UK e-Science All Hands Meeting*, pages 158–165, 2007. [4](#), [48](#), [68](#), [69](#), [71](#), [81](#)
- [AVL⁺11] M. Acosta, M. Esther Vidal, T. Lampo, J. Castillo, and E. Ruckhaus. Anapsid: An adaptive query processing engine for sparql endpoints. In *Proceedings of the 10th International Semantic Web Conference, OSWC'11*, 2011. [3](#), [36](#), [38](#), [45](#), [96](#), [98](#), [117](#)
- [BAAC11] C. Buil-Aranda, M. Arenas, and O. Corcho. Semantics and optimization of the sparql 1.1 federation extension. In *8th Extended Semantic Web Conference (ESWC2011)*, June 2011. [110](#), [111](#), [112](#), [113](#), [116](#)
- [BACK09] C. Buil-Aranda, O. Corcho, and A. Krause. Robust service-based semantic querying to distributed heterogeneous databases. In *Database and Expert Systems Application, 2009. DEXA '09. 20th International Workshop on*, pages 74–78, 31 2009-sept. 4 2009. [5](#)
- [BC07] Christian Bizer and Richard Cyganiak. D2rq - lessons learned. *W3C Workshop on RDF Access to Relational Databases*, October 2007. [44](#)
- [BCGP04] Jesus Barrasa, Oscar Corcho, and Asuncin Gomez-Perez. R2o, an extensible and semantically based database-to-ontology mapping language. In *Proceedings of the 2nd Workshop on Semantic Web and Databases(SWDB2004)*, pages 1069–1070. Springer, 2004. [81](#)
- [BCM⁺10] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 3rd edition, 2010. [12](#)
- [Bec04] D. Beckett. Rdf/xml syntax specification, February 2004. [8](#), [10](#)
- [Bel56] Richard Bellman. Dynamic programming and lagrange multipliers. *Proceedings of the National Academy of Sciences of the United States of America*, 42(10):767–769, 1956. [28](#)
- [BL09] T. Berners-Lee. Design issues: Linked data, 2009. [19](#)
- [BS09] C. Bizer and A. Schultz. The berlin SPARQL benchmark. *International journal on Semantic Web and information systems*, pages 1–24, 2009. [48](#), [97](#), [99](#), [116](#)
- [CCG10] Jean-Paul Calbimonte, Oscar Corcho, and Alasdair J. G. Gray. Enabling ontology-based access to streaming data sources. In *Proceedings of the 9th international semantic web conference on The semantic web - Volume Part I, ISWC'10*, pages 96–111. Springer-Verlag, 2010. [34](#)
- [CFT08] K. G. Clark, L. Feigenbaum, and E. Torres. SPARQL protocol for RDF, January 2008. W3C Recommendation, <http://www.w3.org/TR/rdf-sparql-protocol/>. [1](#)
- [CLNR98] Diego Calvanese, Maurizio Lenzerini, Daniele Nardi, and Riccardo Rosati. Information integration: Conceptual modeling and reasoning support. In *Proceedings of the International Conference on Cooperative Information Systems*, pages 280–291, 1998. [22](#), [23](#)
- [CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009. [28](#)
- [Cza04] Ferguson D. Foster I. Frey J. Graham S. Sedukhin I. Snelling D. Tuecke S. Vambenepe W. Czajkowski, K. The ws-resource framework. Technical report, OASIS, 2004. [72](#), [82](#)
- [DB04] R.V. Guha D. Brickley. Rdf vocabulary description language 1.0: Rdf schema, February 2004. [8](#)
- [DBG⁺] Ewa Deelman, James Blythe, Yolanda Gil, Carl Kesselman, Scott Koranda, Albert Lazzarini, Gaurang Mehta, Maria Papa, and Karan Vahi. Pegasus and the pulsar search: From metadata to execution on the grid. In Roman Wyrzykowski, Jack Dongarra, Marcin Paprzycki, and Jerzy Wasniewski, editors, *Parallel Processing and Applied Mathematics*, Lecture Notes in Computer Science. Springer Berlin / Heidelberg. [68](#)
- [DIR07] Amol Deshpande, Zachary G. Ives, and Vijayshankar Raman. Adaptive query processing. *Foundations and Trends in Databases*, 1(1):1–140, 2007. [35](#)
- [DS04] Mike Dean and Guus Schreiber. OWL web ontology language reference. W3C recommendation, W3C, 2004. [12](#)
- [DS05] M. Durst and M. Suignard. Rfc 3987, internationalized resource identifiers (iris), 2005. [10](#), [53](#)
- [DT10] Krause A. Hume A. C. Grant A. Antonioletti M. Alemu T. Y. Atkinson M. Jackson M. Dobrzelecki, B. and E. Theocharopoulos. Integrating distributed data sources with ogsa-dai dqp and views. *Philosophical Transactions of the Royal Society A*, 2010. [4](#), [68](#), [95](#)
- [ea09] S. Lynden et al. The design and implementation of ogsa-dqp: A service-based distributed query processor. *Future Generation Computer Systems*, 25(3):224–236, 2009. [48](#), [67](#), [69](#)
- [Eke] Joshua Eke. Ogsa-dai parallel hash joins using bonfire. Master's thesis, The University of Edinburgh. [95](#)
- [EP10] C. Buil-Aranda E. Prud'hommeaux. Sparql 1.1 federation extensions, June 2010. [2](#), [53](#), [54](#), [57](#), [58](#)
- [ESW78] Robert Epstein, Michael Stonebraker, and Eugene Wong. Distributed query processing in a relational database system. In *Proceedings of the 1978 ACM SIGMOD international conference on management of data*, pages 169–180. ACM Press, 1978. [30](#)
- [Fie00] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, 2000. AAI9980887. [1](#)
- [FM04] F. Miller F. Manola. Rdf primer, February 2004. [8](#)
- [GHK92] Sumit Ganguly, Waqar Hasan, and Ravi Krishnamurthy. Query optimization for parallel execution. *SIGMOD Rec.*, 21:9–18, June 1992. [29](#)
- [GK04] J. J. Carroll G. Klyne. Resource description framework (rdf): Concepts and abstract syntax, February 2004. [7](#), [8](#)
- [GLR97] Cesar A. Galindo-Legaria and Arnon Rosenthal. Outerjoin simplification and re-ordering for query optimization. *ACM TRANSACTIONS ON DATABASE SYSTEMS*, 22, 1997. [117](#)
- [GO10] B. Glimm and C. Ogbuji. Sparql 1.1 entailment regimes, 2010. [53](#)
- [Gra90] Goetz Graefe. Encapsulation of parallelism in the volcano query processing system. *SIGMOD Rec.*, 19:102–111, May 1990. [29](#), [30](#), [96](#)
- [GS] O. Grlitz and S. Staab. Splendid: Sparql endpoint federation exploiting void descriptions. In *COLD 2011 - Consuming Linked Data Workshop*. [38](#), [39](#), [45](#)
- [GVC11] D. Garijo, B. Villazón, and O. Corcho. A provenance-aware linked data application for trip management and organization. In *7th International Conference on Semantic Systems (iSemantics)*, September 2011. [108](#)
- [GW09] Christine Golbreich and Evan K. Wallace. Owl 2 web ontology language new features and rationale. W3C recommendation, W3C, 2009. [13](#)

REFERENCES

- [Har11] O. Hartig. Zero-knowledge query planning for an iterator implementation of link traversal based query execution. In *Proceedings of the 8th extended semantic web conference on The semantic web: research and applications - Volume Part I*, ESWC'11, pages 154–169, Berlin, Heidelberg, 2011. Springer-Verlag. 39
- [Hay04] P. Hayes. Rdf semantics, February 2004. 8
- [HBF09] O. Hartig, C. Bizer, and JC. Freytag. Executing sparql queries over the web of linked data. In *Proceedings of the 8th International Semantic Web Conference*, ISWC'09, pages 293–309, Berlin, Heidelberg, 2009. Springer-Verlag. 39
- [HD05] Andreas Harth and Stefan Decker. Optimized index structures for querying rdf from the web. In *Proceedings of the Third Latin American Web Congress*, pages 71–, IEEE Computer Society, 2005. 37
- [HF86] Robert Brian Hagmann and Domenico Ferrari. Performance analysis of several back-end database architectures. *ACM Trans. Database Syst.*, 11:1–26, March 1986. 33
- [HH11] Olaf Hartig and Frank Huber. A main memory index structure to query linked data at the world wide web conference (www). In *Proceedings of the 4th Linked Data on the Web (LDOW) Workshop*, 2011. 37, 39
- [HKP⁺09] Pascal Hitzler, Markus Krtzsch, Bijan Parsia, Peter F. Patel-Schneider, and Sebastian Rudolph. OWL 2 web ontology language primer. W3C recommendation, W3C, 2009. 13
- [HS10] S. Harris and A. Seaborne. Sparql 1.1 query language, 2010. 1
- [HSB04] H. Geert-Jan H. Stuckenschmidt, R. Vdovjak and J. Broekstra. Index structures and algorithms for querying distributed rdf repositories. In *WWW*, pages 631–639, 2004. 38
- [Hul97] Richard Hull. Managing semantic heterogeneity in databases: a theoretical perspective. In *Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, PODS '97, pages 51–61, New York, NY, USA, 1997. ACM. 21, 23
- [HWS⁺] Duncan Hull, Katy Wolstencroft, Robert Stevens, Carole A. Goble, Matthew R. Pocock, Peter Li, and Tom Oinn. Taverna: a tool for building and running workflows of services. *Nucleic Acids Research*. 68
- [IC91] Yannis E. Ioannidis and Stavros Christodoulakis. On the propagation of errors in the size of join results. *SIGMOD Rec.*, 20:268–277, April 1991. 35
- [JB03] A. Kampman J. Broekstra. Serql: A second generation rdf query language. In *SWAD-Europe Workshop on Semantic Web Storage and Retrieval*, pages 54–68, 2003. 38
- [JG04] D. Beckett J. Grant. Rdf test cases, February 2004. 8
- [Kos00] Donald Kossmann. The state of the art in distributed query processing. *ACM Computing Surveys*, 32:2000, 2000. 27, 28, 69
- [Lan08] A. Langegger. Virtual data integration on the web: novel methods for accessing heterogeneous and distributed data with rich semantics. In *Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services*, iiWAS '08, pages 559–562, New York, NY, USA, 2008. ACM. 38, 45
- [Lan10] Andreas Langegger. *A Flexible Architecture for Virtual Information Integration Based on Semantic Web Concepts*. PhD thesis, Johannes Kepler University Linz, 2010. 7
- [Len02] Maurizio Lenzerini. Data integration: A theoretical perspective. In *PODS*, pages 233–246, 2002. 21
- [Lev01] Alon Y. Levy. Answering queries using views: A survey. Technical report, VLDB Journal, 2001. 21
- [LKMT11] Steven J. Lynden, Isao Kojima, Akiyoshi Matono, and Yusuke Tanimura. Aderis: An adaptive query processor for joining federated sparql endpoints. In *OTM Conferences* (2), pages 808–817, 2011. 3, 38, 45, 96, 98, 99, 117
- [LS06] T. Banks L. Srinivasan. Web services resource lifetime 1.2, April 2006. 72
- [LSM⁺07] Quanzhong Li, Minglong Shao, Volker Markl, Kevin S. Beyer, Latha S. Colby, and Guy M. Lohman. Adaptively reordering joins during query execution. In *ICDE*, pages 26–35, 2007. 38
- [LT10] Günter Ladwig and Thanh Tran. Linked data query processing strategies. In *Proceedings of the 9th international semantic web conference on The semantic web - Volume Part I*, ISWC'10, pages 453–469, Berlin, Heidelberg, 2010. Springer-Verlag. 37, 38, 96
- [LT11] G. Ladwig and T. Tran. Sihjoin: querying remote and local linked data. In *Proceedings of the 8th extended semantic web conference on The semantic web: research and applications - Volume Part I*, ESWC'11, pages 139–153, Berlin, Heidelberg, 2011. Springer-Verlag. 38, 43, 45, 98, 99
- [MG06] T. Rogers M. Gudgin, M. Hadley. Web services addressing 1.0 - core, May 2006. W3C Recommendation. 72
- [ML86] Lothar F. Mackert and Guy M. Lohman. R* optimizer validation and performance evaluation for distributed queries. In *Proceedings of the 12th International Conference on Very Large Data Bases*, VLDB '86, pages 149–159. Morgan Kaufmann Publishers Inc., 1986. 29
- [MSL10] M. Meier M. Schmidt and G. Lausen. Foundations of sparql query optimization. In *ICDT*, pages 4–33, 2010. 3, 37, 45, 64
- [MSP10] G. Lausen M. Schmidt, T. Hornung and C. Pinkel. Sp2bench: A sparql performance benchmark. In *ICDT*, pages 4–33, 2010. 48, 97, 98, 103, 116
- [NW08] Thomas Neumann and Gerhard Weikum. Rdf-3x: a risc-style engine for rdf. *Proc. VLDB Endow.*, 1:647–659, August 2008. 37
- [PÍ1] Jorge Pérez. *Schema Mapping Management in Data Exchange Systems*. PhD thesis, Pontificia Universidad Católica de Chile, 2011. 26
- [PAG09] J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of SPARQL. *TODS*, 34(3), 2009. 3, 10, 48, 53, 54, 55, 56, 57, 58, 63, 64, 65, 66, 116
- [PKL11] Said Mirza Pahlevi, Isao Kojima, and Steven Lynden. The rdf(s) realization (ws-dairdfs) rdf(s) querying specification. Technical report, OGF DAIS Working Group, 2011. 4
- [PS08] E. Prud'hommeaux and A. Seaborne. SPARQL query language for RDF, January 2008. 1, 2, 53, 54, 86
- [QL08] B. Quilitz and U. Leser. Querying distributed rdf data sources with SPARQL. In *ESWC*, pages 524–538, 2008. 2, 38, 45
- [RNC⁺96] Stuart J. Russell, Peter Norvig, John F. Candy, Jitendra M. Malik, and Douglas D. Edwards. *Artificial intelligence: a modern approach*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996. 12
- [RP06] Jesús Barrasa Rodríguez and Asunción Gómez Pérez. Upgrading relational legacy data to the semantic web. In *Proceedings of the 15th international Conference on World Wide Web*, WWW '06, pages 1069–1070. ACM Press, 2006. 44
- [RRD⁺03] Vijayshankar Raman, Vijayshankar Raman, Amol Deshpande, Amol Deshpande, Joseph M. Hellerstein, and Joseph M. Hellerstein. Using state modules for adaptive query processing. In *ICDE*, pages 353–364, 2003. 36
- [S.95] S. Abiteboul, R. Hull and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995. 119
- [SGH⁺11] Michael Schmidt, Olaf Görlitz, Peter Haase, Günter Ladwig, Andreas Schwarte, and Thanh Tran. Fedbench: a benchmark suite for federated semantic data query processing. In *Proceedings of the 10th international conference on The semantic web - Volume Part I*, ISWC'11, pages 585–600, Berlin, Heidelberg, 2011. Springer-Verlag. 48, 97, 99, 102, 116
- [SHH⁺09] Satya S. Sahoo, Wolfgang Halb, Sebastian Hellmann, Kingsley Idehen, Ted Thibodeau Jr, Sren Auer, Juan Sequeda, and Ahmed Ezzat. A survey of current approaches for mapping of relational databases to rdf. Technical report, W3C RDB2RDF Incubator Group, 2009. 43, 44
- [SHH⁺11] Andreas Schwarte, Peter Haase, Katja Hose, Ralf Schenkel, and Michael Schmidt. Fedx: Optimization techniques for federated query processing on linked data. In *Proceedings of the 10th International Semantic Web Conference ISWC 2011*, 2011. 2, 38, 39, 45
- [SL90] Amit P. Sheth and James A. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Comput. Surv.*, 22:183–236, September 1990. ix, 22, 24, 25, 26
- [SMK97] Michael Steinbrunn, Guido Moerkotte, and Alfons Kemper. Heuristic and randomized optimization for the join ordering problem. *The VLDB Journal*, 6:191–208, August 1997. 30

REFERENCES

- [SS08] S. Schenk and S. Staab. Networked graphs: a declarative mechanism for SPARQL, rules, SPARQL views and rdf data integration on the web. In *WWW*, pages 585–594, 2008. 2, 38, 45
- [TB06] J. Treadwell T. Banks. Web services resource properties 1.2. OASIS standard, April 2006. 72
- [TBL05] L. Masinter T. Berners-Lee, R. Fielding. Rfc 3986 uniform resource identifier (uri): Generic syntax. <http://www.ietf.org/rfc/rfc3986.txt>, January 2005. 7
- [UF00] Tolga Urhan and Michael J. Franklin. Xjoin: A reactively-scheduled pipelined join operator. *IEEE DATA ENGINEERING BULLETIN*, 23:2000, 2000. 36
- [UFA98] Tolga Urhan, Michael J. Franklin, and Laurent Amsaleg. Cost-based query scrambling for initial delays. In *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, SIGMOD '98, pages 130–141, New York, NY, USA, 1998. ACM. 36
- [Ull97] Jeffrey D. Ullman. Information integration using logical views. In *Proceedings of the 6th International Conference on Database Theory*, pages 19–40, London, UK, 1997. Springer-Verlag. 21
- [VRL⁺10] Maria-Esther Vidal, Edna Ruckhaus, Tomas Lampo, Amads Martnez, Javier Sierra, and Axel Polleres. Efficiently joining group patterns in sparql queries. In *ESWC (1)'10*, pages 228–242, 2010. 36
- [Wie92a] Gio Wiederhold. Mediators in the architecture of future information systems. *Computer*, 25:38–49, March 1992. 21
- [Wie92b] Gio Wiederhold. Mediators in the architecture of future information systems. *Computer*, 25:38–49, March 1992. 93
- [WKB08] Cathrin Weiss, Panagiotis Karras, and Abraham Bernstein. Hexastore: sextuple indexing for semantic web data management. *Proc. VLDB Endow.*, 1:1008–1019, August 2008. 37