# A Data-flow Language for Big RDF Data Processing

Fadi Maali [⋆]

Insight Centre for Data Analytics, National University of Ireland Galway
`fadi.maali@insight-centre.org`

**Abstract.** When analysing large RDF datasets, users are left with two main options: using SPARQL or using an existing non-RDF-specific big data language, both with its own limitations. The pure declarative nature of SPARQL and the high cost of evaluation can be limiting in some scenarios. On the other hand, existing big data languages are designed mainly for tabular data and, therefore, applying them to RDF data results in verbose, unreadable, and sometimes inefficient scripts. My PhD work aims at enhancing programmability of big RDF data. The gaol is to augment the existing tools with a declarative dataflow language that focuses on the analysis of large-scale RDF data. Similar to other big data processing languages, I aim at identifying a set of basic operators that are amenable to parallelisation and at supporting extensibility via user-defined custom code. On the other hand, a graph-based data model and support for pattern matching as in SPARQL are to be adopted. Giving the focus on large-scale data, scalability and efficiency are critical requirements. In this paper, I report on my research plan and describe some preliminary results.

## 1 Problem Statement

Petabytes and terabytes datasets are becoming commonplace, especially in industries such as telecom, health care, retail, pharmaceutical and financial services. To process these huge amounts of data, a number of distributed computational frameworks have been suggested recently [7, 13, 31]. Furthermore, there has been a surge of activity on layering declarative languages on top of these platforms. Examples include Pig Latin from Yahoo [16], DryadLINQ from Microsoft [30], Jaql from IBM [3], HiveQL from Facebook [27] and Meteor/Sopremo [11].

In the Semantic Web realm, this surge of analytics languages was not reflected despite the significant growth in the available RDF data. To analyse large RDF datasets, users are left mainly with two options: using SPARQL [10] or using an existing non-RDF-specific big data language. I argue that each of these options has its own limitations.

SPARQL is a graph pattern matching language that provides rich capabilities for slicing and dicing RDF data. The latest version, SPARQL 1.1, supports

[⋆] Supervisor: Prof. Dr. Stefan Decker. Expected graduation date: Fall 2016 also aggregation and nested queries. Nevertheless, the pure declarative nature of SPARQL obligates a user to express their needs in a single query. This can be unnatural for some programmers and challenging for complex needs [15, 9]. Furthermore, SPARQL evaluation is known to be costly [18, 23] and requires all data to be transformed into RDF beforehand.

The other alternative of using an existing big data language such as Pig Latin or HiveQL has also its own limitations. These languages were designed for tabular data mainly, and, consequently, using them with RDF data commonly results in verbose, unreadable, and sometimes inefficient scripts [21].

My PhD work aims at enhancing **programmability of big RDF data**. The gaol is to augment the existing tools with a declarative dataflow language that focuses on the analysis of large-scale RDF data. Similar to other big data processing languages, I aim at identifying a small set of basic operators that are amenable to parallelisation and at supporting extensibility via user-defined custom code. On the other hand, a graph-based data model and support for pattern matching as in SPARQL are to be adopted. Giving the focus on large-scale data, scalability and efficiency are critical requirements. Moreover, I intend to work towards relaxing the prerequisite of full transformation of non-RDF data and facilitating processing of RDF and non-RDF data together.

## 2 Relevancy

Data is playing a crucial role in societies, governments and enterprises. For instance, data science is increasingly utilised in supporting data-driven decisions and in delivering data products [14, 20]. Furthermore, scientific fields such as bioinformatics, astronomy and oceanography are going through a shift from "querying the world" to "querying the data" in what commonly referred to as e-science [12]. The main challenge nowadays is analysing the data and extracting useful insights from it.

Declarative languages simplify programming and reduce the cost of creation, maintenance, and modification of software. They also help bringing the non-professional user into effective communication with a database [5]. In 2008, the Claremont Report on Database Research identified declarative programming as one of the main research opportunities in the data management field [1].

My PhD work intends to facilitate analysing large amount of RDF data by designing a declarative language. The fast pace at which the data is growing and the expected shortage in people with data analytical skills [6], make users' productivity of paramount importance. Moreover, By embracing the process of RDF and non-RDF data together, my hope is to increase the utilisation of the constantly growing size of RDF data.

## 3 Related Work

A large number of declarative languages were introduced recently as part of the big data movement. These languages vary in their programming paradigm, and in their underlying data model. Pig Latin [16] is a dataflow language with a tabular data model that also supports nesting. Jaql [3] is a declarative scripting language that blends in a number of constructs from functional programming languages and uses JSON for its data model. HiveQL [27] adopts a declarative syntax similar to SQL and its underlying data model is a set of tables. Other examples of languages include Impala[1], Cascalog[2], Meteor [11] and DryadLINQ [30]. [26] presented a performance as well as language comparison of HiveQL, Pig Latin and Jaql. [22] also compared a number of big data languages but focuses on their compilation into a series of MapReduce jobs.

In the semantic web field, SPARQL is the W3C recommended querying language for RDF. A number of extensions to SPARQL were proposed in the literature to support search for semantic associations [2], and to add nested regular expressions [19] for instances. However, these extensions do not change the pure declarative nature of SPARQL. There are also a number of non-declarative languages that can be integrated in common programming languages to provide support for RDF data manipulation [17, 25]. In the more general context of graph processing languages, [29] provides a good survey.

## 4 Research Questions

A core part of a declarative language is its underlying data model. A data model consists of a notation to describe data and a set of operations used to manipulate that data [28]. SPARQL Algebra [18] is the data model underlying SPARQL. SPARQL Algebra cannot be used as an underlying model for the declarative language I am working on for the following reasons:

- It is not fully composable. The current SPARQL algebra transitions from graphs (i.e. the initial inputs) to sets of bindings (which are basically tables resulting from pattern matching). Subsequently, further operators such as Join, Filter, and Union are applied on sets of bindings. In other words, the flow is partly “hard-coded” in the SPARQL algebra and a user cannot, for instance, apply a pattern matching on the results of another pattern matching or “join” two graphs. In a dataflow language, the dataflow is guided by the user and cannot be limited to the way SPARQL Algebra imposes.
- It assumes all data is in RDF.
- The expressivity of SPARQL comes at the cost of high evaluation complexity [18, 23].

Therefore, the main challenge is to define an adequate data model that embraces RDF and non-RDF data and strikes a balance between expressivity and complexity. Accordingly, my research questions are:

**RQ1:** What is the appropriate data model to adopt?
**RQ2:** How do we achieve efficient scalable performance?
**RQ3:** How do we enable processing of RDF and non-RDF data together?

[1] `https://github.com/cloudera/impala`
[2] `http://cascalog.org/`

## 5 Hypotheses

We introduce several hypotheses that we would like to test in our research.

**H1:** A new data model can be defined to underlie a dataflow language for RDF data. The expressivity and complexity of this data model can be determined.
**H2:** Algebraic properties of the new data model can be exploited to enhance performance.
**H3:** Scalable efficient performance can be achieved by utilising state-of-the-art distributed computational frameworks.
**H4:** Integrating transformation to RDF as part of the data processing enables processing RDF and non-RDF data together and can eliminate the need of *full* transformation to RDF.

## 6 Approach & Preliminary Results

We had an initial proposal for a data model and a dataflow language. Our goal is to iteratively refine the model **(H1, H2)** and our implementation **(H3)** and then extend it to include non-RDF data and data transformation **(H4)**. The next two subsections summarise our preliminary results.

### 6.1 RDF Algebra

RDF Algebra is our proposed data model. This algebra defines operators similar to those defined in SPARQL algebra but that are fully composable. To achieve such composability, the algebra operators input and output are always a pair of a graph and a corresponding table **(H1)**. The core set of expressions in this algebra are: atomic, projection, extending, triple pattern matching, filtering, cross product and aggregation. The syntax and the semantics of these expressions have been formally defined and their expressivity in comparison to SPARQL is captured by the following lemma.

**Lemma 1.** *RDF Algebra expressions can express SPARQL 1.1 basic graph patterns with filters, aggregations and assignments.*

We have also started to study some unique algebraic properties of our data model **(H2)**. Cascading triple patterns and joins in the RDF algebra results in some unique optimisation opportunities. Therefore, we defined a partial ordering relationship between triple patterns to capture subsumption among results. Consequently, evaluation plans can be optimised and intermediary results can be reused in order to enhance evaluation performance **(H3)**.

The innovative part of this model is the pairing of graphs and tables, which, to the best of our knowledge, was not reported in the literature before. This ensures full composability and can potentially accommodate tabular data (with empty graph component that can be populated via transforming the tabular data only when necessary) **(H4)**.

## 6.2 SYRql, A Dataflow Language

Our current dataflow language that is grounded in the algebra defined before is called SYRql. A SYRql script is a sequence of statements and each statement is either an assignment or an expression. The syntax of SYRql borrows the use of "$->$" syntax from Jaql to explicitly show the data flow. Whereas pattern matching in SYRql uses identical syntax to basic graph patterns of SPARQL. SPARQL syntax for patterns is intuitive, concise and well-known to many users in the Semantic Web field. We hope that this facilitates learning SYRql for many users.

The current implementation[3] uses JSON[4] for internal representation of the data. Particularly, we use JSON arrays for bindings and JSON-LD [24] to represent graphs. SYRql scripts are parsed and then translated into a directed acyclic graph (DAG) of MapReduce jobs **(H3)**. Sequences of expressions that can be evaluated together are grouped into a single MapReduce job. Finally, the graph is topologically sorted and the MapReduce jobs are scheduled to execute on the cluster. Our initial performance evaluation showed comparative performance to well-established languages such as Pig Latin and Jaql (Figure 1).

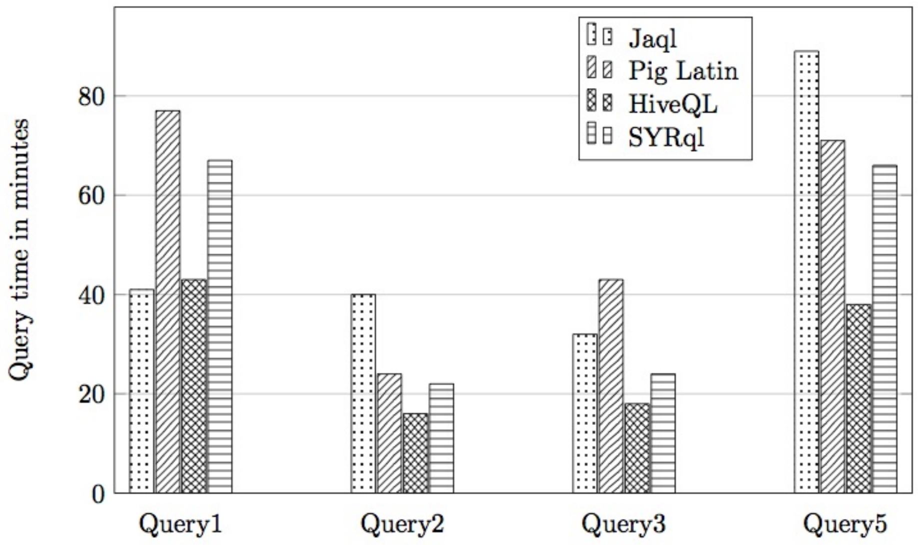

**Fig. 1.** Query processing times

[3] https://gitlab.deri.ie/Maali/syrql-jsonld-imp/wikis/home
[4] http://json.org

## 7 Evaluation Plan

We are currently conducting a formal study of the data model, its algebraic properties, its complexity and expressivity. We plan to compare it to First-Order Logic languages **(H1, H2)**.

For performance evaluation, we have started comparing response time that our implementation provides to that of SPARQL and existing big data languages **(H3)**. Figure 1 shows initial results. The benchmark we used is based on the Berlin SPARQL Benchmark (BSBM) [4] that defines an e-commerce use case. Specifically, we translated a number of queries in the BSBM Business Intelligence usecase (BSBM BI)[5] into equivalent programs in HiveQL, Pig Latin and Jaql. To the best of our knowledge, this is the first benchmark that uses existing big data languages with RDF data.

Furthermore, we plan to use some data manipulation scenarios from bioinformatics research to guide requirement collection for processing RDF and non-RDF data **(H4)**. We plan to conduct a performance evaluation and a user study to evaluate our work on this regards.

## 8 Reflections

We base our work on a good understanding of Semantic Web technologies as well as existing Big Data techniques and languages. The initial results we have collected are promising. Nevertheless, the current implementation leaves rooms for improvements. We plan to use RDF compression techniques such as HDT [8] and to experiment with distributed frameworks other than MapReduce such as Spark. Finally, we believe that our data model and its algebraic properties can yield fruitful results that can further be applied in tasks like caching RDF query results, views management and query results reuse.

## References

1. Rakesh Agrawal et al. The Claremont Report on Database Research. *SIGMOD Rec.*, 2008.
2. Kemafor Anyanwu and Amit Sheth. P-queries: enabling querying for semantic associations on the semantic web. In *WWW*, 2003.
3. Kevin S. Beyer, Vuk Ercegovac, Rainer Gemulla, Andrey Balmin, Mohamed Y. Eltabakh, Carl-Christian Kanne, Fatma Özcan, and Eugene J. Shekita. Jaql: A Scripting Language for Large Scale Semistructured Data Analysis. *PVLDB*, 2011.
4. Christian Bizer and Andreas Schultz. The Berlin SPARQL Benchmark. *IJSWIS*, 2009.
5. Donald D. Chamberlin and Raymond F. Boyce. SEQUEL: A Structured English Query Language. In *SIGFIDET*, 1974.

[5] `http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/spec/BusinessIntelligenceUseCase/index.html`

6. Michael Chui, Brad Brown, Jacques Bughin, Richard Dobbs, Charles Roxburgh, and Angela Hung Byers James Manyika. Big data: The next frontier for innovation, competition, and productivity. Technical report, McKinsey Global Institute, 2011.
7. Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *OSDI*, 2004.
8. Javier D Fernández, Miguel A Martínez-Prieto, Claudio Gutiérrez, Axel Polleres, and Mario Arias. Binary rdf representation for publication and exchange (hdt). *Web Semantics: Science, Services and Agents on the World Wide Web*, 19:22–41, 2013.
9. Stefan Hagedorn and Kai-Uwe Sattler. Efficient Parallel Processing of Analytical Queries on Linked Data. In *OTM*, 2013.
10. Steve Harris and Andy Seaborne. SPARQL 1.1 Query Language. W3C Recommendation 21 March 2013. http://www.w3.org/TR/sparql11-query/.
11. Arvid Heise, Astrid Rheinländer, Marcus Leich, Ulf Leser, and Felix Naumann. Meteor/Sopremo: An Extensible Query Language and Operator Model. In *BigData*, 2012.
12. Tony Hey, Stewart Tansley, and Kristin Tolle, editors. *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Microsoft Research, 2009.
13. Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: Distributed Data-parallel Programs from Sequential Building Blocks. In *EuroSys*, 2007.
14. Mike Loukides. What is Data Science? *O'Reilly radar*, 6 2010.
15. Fadi Maali and Stefan Decker. Towards an RDF Analytics Language: Learning from Successful Experiences. In *COLD*, 2013.
16. Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. Pig Latin: a Not-so-foreign Language for Data Processing. In *SIGMOD*, 2008.
17. Eyal Oren, Renaud Delbru, Sebastian Gerke, Armin Haller, and Stefan Decker. Activerdf: Object-oriented semantic web programming. In *WWW*, 2007.
18. Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and Complexity of SPARQL. ISWC, 2006.
19. Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. nSPARQL: A navigational language for RDF. *Web Semantics: Science, Services and Agents on the World Wide Web*, 2010.
20. Foster Provost and Tom Fawcett. Data Science and its Relationship to Big Data and Data-Driven Decision Making. *Big Data*, 1(1), March 2013.
21. Padmashree Ravindra, HyeongSik Kim, and Kemafor Anyanwu. An Intermediate Algebra for Optimizing RDF Graph Pattern Matching on MapReduce. In *ESWC*, 2011.
22. Caetano Sauer and Theo Haerder. Compilation of query languages into mapreduce. *Datenbank-Spektrum*, 2013.
23. Michael Schmidt, Michael Meier, and Georg Lausen. Foundations of sparql query optimization. In *ICDT*, 2010.
24. Manu Sporny, Dave Longley, Gregg Kellogg, Markus Lanthaler, and Niklas Lindström. JSON-LD 1.0. W3C Recommendation 16 January 2014.
25. Steffen Staab. Liteq: Language integrated types, extensions and queries for rdf graphs. *Interoperation in Complex Information Ecosystems*, 2013.
26. Robert J Stewart, Phil W Trinder, and Hans-Wolfgang Loidl. Comparing High Level MapReduce Query Languages. In *APPT*. 2011.

27. Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Ning Zhang 0002, Suresh Anthony, Hao Liu, and Raghotham Murthy. Hive - a Petabyte Scale Data Warehouse Using Hadoop. In *ICDE*, 2010.
28. J.D. Ullman. *Principles of Database and Knowledge-base Systems*, chapter 2. Computer Science Press, Rockville, 1988.
29. Peter T. Wood. Query Languages for Graph Databases. *SIGMOD*, 2012.
30. Yuan Yu, Michael Isard, Dennis Fetterly, Mihai Budiu, Úlfar Erlingsson, Pradeep Kumar Gunda, and Jon Currey. DryadLINQ: a System for General-purpose Distributed Data-parallel Computing Using a High-level Language. In *OSDI*, 2008.
31. Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *NSDI*, 2012.