Interroger des Lacs de Données en utilisant Spark & Presto

Mohamed Nadjib Mami†, Damien Graux‡, Simon Scerri§, Hajira Jabeen§, and Sören Auer‡
† Fraunhofer IAIS; ‡ University of Bonn
# ADAPT Centre, Trinity College Dublin; £ TIB and L3S Research Center
\{mami,scerri,jabeen\}@cs.uni-bonn.de; damien.graux@adaptcentre.ie; auer@l3s.de

ABSTRACT
Squerall est un outil permettant l’interrogation de sources de données hétérogènes à large échelle en utilisant à bon escient des moteurs de traitement dédiés aux larges volumes de données issus de la littérature: Spark et Presto. Les requêtes à destination des lacs de données sont évaluées à la volée, i.e., directement sur les sources originelles sans procéder à de quelconques transformations préalables des données. Nous démontrons la capacité qu’a Squerall à interagir avec cinq sources différentes parmi lesquelles Cassandra et MongoDB. En particulier, nous mettons en évidence que notre outil peut joindre ensemble plusieurs sources en même temps, tout en montrant qu’étendre la couverture à d’autres sources potentielles reste simple. Des interfaces graphiques sont aussi mises à disposition pour (1) construire les requêtes SPARQL et (2) mettre en place les fichiers de configuration nécessaires.

CCS CONCEPTS
• Information systems → Database query processing; Parallel and distributed DBMSs; Mediators and data integration;
• Applied computing → Information integration and interoperability; • Computing methodologies → Knowledge representation and reasoning.

1 INTRODUCTION
During the last four decades, a variety of data storage and management techniques have been developed in both research and industry. Today, we benefit from a multitude of storage solutions, varying in their data model (e.g. tabular, document, graph) or their ability to scale storage and querying. There are dozens of continuously evolving storage and data management solutions. As a result, users can choose a system that suits their individual application needs. However, those systems do not inter-operate, every stored datum is locked in the respective system it is stored in. For example, an e-commerce company might store product information in a Cassandra database, offers in MongoDB to benefit from its capability to store hierarchical multi-level values, and information about Producers obtained from an external source in a relational format. Without transforming and moving the data into a unified (scalable) data management solution, the data can hardly be explored and business insights be extracted using ad hoc uniform querying. We have taken on the mission of bridging this gap and developed Squerall\(^1\) [9]: a software that gives access to heterogeneous data kept in their original forms and sources using Semantic Web techniques to enable uniform querying with SPARQL\(^2\).

Similar efforts to integrate and query large data sources exist in the literature. For instance, [3] defines a mapping language to express access links to NoSQL databases. [11] allows to run CRUD operations over NoSQL databases. [1] proposes a unifying programming model to directly access databases using get, put and delete primitives. [7] proposes a SQL-like language containing invocations to the native query interface of relational and NoSQL databases. [6] is a hybrid platform with consideration for both heterogeneous and dynamic data sources (streams). However, Squerall offers the highest number of supported data sources (namely: CSV, Parquet, Cassandra, MongoDB and MySQL) while providing the richest query capability, including joining, aggregation and ordering.

2 SQUERALL: CONCEPTS & ARCHITECTURE
Squerall [9] implements the so-called Ontology-Based Data Access (OBDA) [10] paradigm. In OBDA, data schemata are mapped to higher-level ontologies, forming a middleware against which queries are posed. These SPARQL queries are then executed in a separate distributed environment, which is, in particular, resilient to faults (node failure does not halt the entire query execution), and elastic and horizontally-scalable (more nodes can be added to accommodate more expensive computations). In addition, as data from different sources is generated by different applications, they are

\(^1\)https://eis-bonn.github.io/Squerall/
\(^2\)http://www.w3.org/TR/sparql11-overview/
may not be able to be readily cross-joined. Thus, modifications on the possible join values ought to be incorporated. Squerall is comprised of five components (see Figure 1):

- **Query Decomposer**: Validates and analyzes SPARQL queries provided by a user. Particularly, the Query Decomposer extracts the Basic Graph Pattern fragment of the query and decomposes it into star-shaped sub-graph patterns having the same subject, stars for short. This component also detects links between stars.

- **Relevant Entity Extractor**: Each star is analyzed separately; this component searches in the mappings for entities that are mapped to every predicate of the star.

- **Data Connector**: Once relevant data entities are detected, they are connected to the distributed execution environment. Every detected entity is loaded into a ParSet (Parallel dataSet): a data structure that can be distributed and operated on concurrently. The Data Connector expects users to input connection metadata.

- **Distributed Query Processor**: Following the principles introduced earlier, queries are executed in parallel. Query execution occurs on and across the ParSets. Links between stars retrieved by the Query Decomposer are transformed into joins between the relevant detected data entities and all stars are incrementally joined. When disjoiunty points are known, join values are altered to enable joinability.

- **Query Designer**: We see the necessity of supporting users in their SPARQL query creation.

### 3 TECHNICAL DETAILS

#### Core technologies

RML [5] and FNO [4] are used to express mappings and to declare query-independent transformations. Apache Spark and Presto are used to implement both the Data Connector and Distributed Query Processor. Spark is a general-purpose processing engine, and Presto is a distributed SQL query engine. Both provide dozens of wrappers to connect to a data source. They load the data (fully, or partially) to their in-memory data structures.

#### Achieved Performance

We evaluate the performance querying five different data sources: Cassandra, MySQL, MongoDB, Parquet, and CSV. As evaluation data we choose the BSBM [2] benchmark. We pick five relational tables (Product, Producer, Offer, Review, and Person) and load them into the five data sources. For performance and scalability we evaluate the execution time of BSBM queries against three increasing data sizes: 0.5m, 1.5m and 5m scale factor of 0.5 millions products – scale factor of 1.5 millions products – scale factor of 0.5 millions products.

#### Table 1: Query execution times (seconds) using Presto and Spark and the difference percentage between them (%)

<table>
<thead>
<tr>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Q4</th>
<th>Q5</th>
<th>Q6</th>
<th>Q7</th>
<th>Q8</th>
<th>Q9</th>
<th>Q10</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Presto</strong></td>
<td>55.34</td>
<td>28.89</td>
<td>15.84</td>
<td>53.63</td>
<td>49.24</td>
<td>43.58</td>
<td>18.63</td>
<td>14.58</td>
<td>89.08</td>
</tr>
<tr>
<td><strong>Spark</strong></td>
<td>96.78</td>
<td>189.57</td>
<td>59.96</td>
<td>277.30</td>
<td>222.76</td>
<td>191.26</td>
<td>159.51</td>
<td>91.58</td>
<td>360.39</td>
</tr>
<tr>
<td><strong>Diff. %</strong></td>
<td>71.44</td>
<td>160.68</td>
<td>44.12</td>
<td>227.47</td>
<td>173.52</td>
<td>142.69</td>
<td>102.73</td>
<td>76.99</td>
<td>271.51</td>
</tr>
<tr>
<td><strong>Spark</strong></td>
<td>276.91</td>
<td>434.05</td>
<td>504.41</td>
<td>552.40</td>
<td>526.93</td>
<td>1060.83</td>
<td>523.31</td>
<td>219.05</td>
<td></td>
</tr>
<tr>
<td><strong>Presto</strong></td>
<td>178.48</td>
<td>378.57</td>
<td>517.06</td>
<td>452.40</td>
<td>400.88</td>
<td>856.31</td>
<td>635.33</td>
<td>337.21</td>
<td></td>
</tr>
<tr>
<td><strong>Diff. %</strong></td>
<td>47.80</td>
<td>656.17</td>
<td>378.57</td>
<td>517.06</td>
<td>452.40</td>
<td>400.88</td>
<td>856.31</td>
<td>635.33</td>
<td>337.21</td>
</tr>
</tbody>
</table>

#### User Interfaces

We provide 3 GUIs to help users produce Squerall’s inputs: Config, Mappings files and SPARQL query. They have built-in search functionalities that send requests to the LOV catalog to search for adequate terms from existing ontologies.

### 4 CONCLUSION

Squerall addresses the Variety challenge of Big Data by making use of Semantic Web standards and best practices. It can be extended to embrace new data sources, by making use of the query engines’ own wrappers. Additionally, Squerall has been integrated into SANS [8], a framework for scalable processing and analysis of large-scale RDF data, widening its scope to also access non-RDF data sources. Squerall source code is available under an Apache-2.0 license on GitHub. In addition, a screencast presenting the various interfaces and the query execution is available. The deployment is further facilitated with a Dockerfile to quickly run the BSBM use-case described here.

#### REFERENCES


*Open O熠Vocabularies: publish and search ontologies [https://lov.linkeddata.es/]

*GitHub SANS Stack: [https://github.com/EIS-Bonn/Sanes-Stack](https://github.com/EIS-Bonn/Sanes-Stack)

*GitHub SANS Stack: [https://github.com/EIS-Bonn/Sanes-Stack](https://github.com/EIS-Bonn/Sanes-Stack)

*GitHub SANS Stack: [https://github.com/EIS-Bonn/Sanes-Stack](https://github.com/EIS-Bonn/Sanes-Stack)