RDF triplestores and SPARQL endpoints

Lecturer: Mathias Bonduel
mathias.Bonduel@kuleuven.be

LDAC summer school 2019 – Lisbon, Portugal
Lecture outline

• **Storing** RDF data: **RDF triplestores**
  - Available methods to store RDF data
  - RDF triplestores
  - Triplestore applications – databases – default graph – named graphs
  - List of triplestore applications
  - Comparing triplestores
  - Relevant triplestore settings
  - Communication with triplestores

• **Distributing** RDF data: **SPARQL endpoints**
  - Available methods to distribute RDF data
  - SPARQL endpoints
  - Reuse of SPARQL queries
  - SPARQL communication protocol: requests and responses
Storing RDF data
Available methods to store RDF data

• In-memory storage (local RAM)
  o Working memory of application (e.g. client side web app, desktop app)
  o Frameworks/libraries: RDFLib (Python), rdflib.js (JavaScript), N3 (JavaScript), rdfstore-js (JavaScript), Jena (Java), RDF4J (Java), dotNetRDF (.NET), etc.
  o Varied support for SPARQL querying

• Persistent storage (storage drive)
  o RDF file/dump (diff. RDF serializations): TTL, RDF/XML, N-Quads, JSON-LD, N-triples, TriG, N3, TriX, RDFa (RDF embedded in HTML), etc.
  o RDF triplestore
  o (ontology editing applications: Protégé, Topbraid Composer, etc.)
RDF triplestores

“a database to store and query RDF triples”

- Member of the family of graph/NoSQL databases
- Data structure: RDF
- Main query language: SPARQL
- Oftentimes support for RDFS/OWL/rules reasoning
- Data storage is typically persistent
Triplestore applications – databases - default graph - named graphs

• An RDF triplestore instance (application) can have one or multiple databases (repositories)

• Each database has one default graph and zero or more named graphs
  o a good practice is to place TBox in a separate named graph. Most triplestores’ reasoning engines consider all TBox statements in the database
  o You can also use the TBox without reasoning => query directly
Instance of an RDF triplestore application (e.g. an Apache Jena Fuseki instance)
List of triplestore applications

- Apache Jena:
  - TDB
  - Fuseki
- Blazegraph DB
- Stardog
- Strabon
- Apache Marmotta/KiWi
- Ontotext GraphDB (former OWLIM)
- Openlink Virtuoso
- Cambridge Semantics AnzoGraph
- Amazon Neptune
- Franz AllegroGraph
- BBN Parliament
- ClioPatria
- ...

June 18, 2019
RDF triplestores and SPARQL endpoints | Mathias Bonduel
8
List of triplestore applications

• Apache Jena Fuseki
• Stardog
• Ontotext GraphDB
• BBN Parliament
Comparing triplestores (1/4)

- **Hardware** sizing: recommended RAM, disk space, etc.
- **Efficiency** (speed and memory usage)
  - RDF loading/import efficiency
  - Querying efficiency
  - Reasoning efficiency
- **License** options
  - open source/proprietary
  - free/educational/commercial
- **Security**: SSH, access roles
Comparing triplestores (2/4)

- **API** programming language support
- **SPARQL 1.1** support + SPARQL extensions (spatial, temporal, PATHS, etc.)
- **Reasoning**: support for RDFS, OWL flavors, rule languages, SHACL
- **Validity/conformity** check of input data
- **Documentation** (or the lack of it)
- **(Commercial) support**
- **Extras:**
  - GraphQL, support for Apache TinkerPop API (e.g. Gremlin), etc.
  - plug-and-play cloud database
  - full-text search, machine learning
  - virtual graphs
Comparing triplestores (3/4)

<table>
<thead>
<tr>
<th>Name triplestore</th>
<th>GeoSPARQL querying</th>
<th>Multiple repositories per DB instance</th>
<th>Named graphs</th>
<th>Supported reasoning schemes</th>
<th>Licensing options</th>
<th>Bulk loading</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache Jena Fuseki</td>
<td>No*</td>
<td>Yes</td>
<td>Yes</td>
<td>RDFS/OWL rule engines**</td>
<td>Open source (Apache License v2.0)</td>
<td>Yes</td>
</tr>
<tr>
<td>Stardog</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>RDFS, OWL DL/QL/RL, SL (includes SWRL), SHACL</td>
<td>Proprietary: Enterprise licenses (60 day trial), free academic license</td>
<td>Yes</td>
</tr>
<tr>
<td>Ontotext GraphDB</td>
<td>No***</td>
<td>Yes</td>
<td>Yes</td>
<td>RDFS, OWL QL/RL, custom rulesets</td>
<td>Proprietary: Enterprise license (60 day trial), standard license (60 day trial), free license</td>
<td>Yes</td>
</tr>
<tr>
<td>BBN Parliament</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Selection of RDFS and OWL rules (config), SWRL</td>
<td>Open source (BSD)</td>
<td>Yes</td>
</tr>
</tbody>
</table>

* Limited custom geospatial queries possible over WGS84 and GeoSPARQL WKT data (newest versions might include GeoSPARQL querying)
** Jena modules can be connected for extensions: [http://jena.apache.org/documentation/inference/](http://jena.apache.org/documentation/inference/)
*** Limited custom geospatial queries possible over WGS84 data (GraphDB functions)

Disclaimer: no liability regarding completeness and correctness

June 18, 2019
RDF triplestores and SPARQL endpoints | Mathias Bonduel
## Comparing triplestores (3/4)

<table>
<thead>
<tr>
<th>Name</th>
<th>Support</th>
<th>Programming interface</th>
<th>Extras: rule languages, validity check, built-in security</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache Jena</td>
<td>(Public mailinglist)</td>
<td>Java (Jena)</td>
<td>Support for rules, authentication (Apache Shiro), extensions via Jena (advanced reasoning, geospatial querying, full text search, etc.)</td>
</tr>
<tr>
<td>Fuseki</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stardog</td>
<td>Commercial support, Stardog forum</td>
<td>Java (SNARL (native API), Sesame, RDF4J, Jena)</td>
<td>Validation (SHACL/ICV), rules (SWRL and Stardog Rules), PATHS queries, virtual graphs (RDBMS, CSV/JSON/XML, MongoDB, Cosmos DB, Cassandra, etc. using R2RML or Stardog mapping syntax), GraphQL, property graph model (Tinkerpop), built-in authentication, full-text search, NLP, query/reasoning(inconsistency/inference)/validation explanations, machine learning functions</td>
</tr>
<tr>
<td>Ontotext</td>
<td>Commercial support, email, Stack Overflow</td>
<td>Java (RDF4J, Jena)</td>
<td>Validity check, separate plug and play cloud database, full-text search, RDFRank (interconnectedness), built-in authentication, custom GraphDB Rules, query explanations</td>
</tr>
<tr>
<td>GraphDB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BBN Parliament</td>
<td>(Github)</td>
<td>Java, C++</td>
<td>Support for SWRL rules, temporal indexing</td>
</tr>
</tbody>
</table>

Disclaimer: no liability regarding completeness and correctness
Relevant triplestore configuration settings (1/2)

• SPARQL endpoint host and port
• Query timeout
• **Heap size**
• Security and authorisation
  - Connection methods to HTTP server: enable/require SSL encryption
  - users: name and password
  - roles connected to users
  - rights for user/role: read/write RDF content, create/delete databases, adjust settings/users, grant/revoke permissions
• Treatment of literals: strict parsing, canonicalization

<table>
<thead>
<tr>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>JVM Heap</td>
</tr>
<tr>
<td>Off-Heap</td>
</tr>
<tr>
<td>OS</td>
</tr>
</tbody>
</table>

*Settings* with potential impact on loading/querying efficiency
Relevant triplestore configuration settings (1/2)

• **Indexing**: regular, geospatial, temporal, text (literals)
• Query scope: default graph, union of all named graphs, union of all named graphs and default graph
• Active **reasoning** profiles: RDFS, OWL RL, OWL QL, etc.
• Active **rule** engines
• Enable spatial querying, reasoning, etc.
• ...

**Settings** with potential impact on loading/querying efficiency
Communication with triplestores

• SPARQL over HTTP request: **SPARQL protocol** (code, cURL, Postman, etc.):
  o SPARQL endpoint URL: read ⇔ update
  o Options: query, authentication, inference, input/output format
  o W3C® standardized

• Native API of triplestore (own code, commandline):
  o Bulk loading RDF files
  o Management of databases: (all/active) databases, start, stop, drop, settings, roles, etc.
  o Direct queries

• Web/desktop client:
  o Triplestore specific client: via SPARQL protocol or native API
  o Generic client: via SPARQL protocol
Links to installation guidelines

• Apache Jena Fuseki:
  o Visual interface: Fuseki web app on localhost:3030 (comes with triplestore)
  o Triplestore: all OS

• Stardog:
  o Visual interface: Stardog Studio (and web app)
  o Triplestore: Linux and Mac OS / Docker / Windows

• GraphDB:
  o Visual interface: GraphDB workbench web app on localhost:7200 (comes with triplestore)
  o Triplestore: Linux / Mac OS / Windows

• Parliament:
  o Visual interface: Parliament web app on localhost:8089/parliament (comes with triplestore)
  o Triplestore: all OS (download from Github repository, old website is depreciated)
Distributing RDF data
Available methods to distribute RDF data

- SPARQL endpoint
- Triple Pattern Fragment server endpoint
- RDF files:
  - dump file
  - subject page
  - RDF embedded in HTML (i.e. RDFa)

in-memory RDF store (local access only)

Source: http://linkeddatafragments.org/concept/
SPARQL endpoint (1/4)

“An entry point for HTTP access to shared RDF data using SPARQL as a query language”

(standardized SPARQL communication protocol)
SPARQL endpoint (2/4)

HTTP request with SPARQL query

HTTP response

Client

Server

Triple stores

RDF files

RDBMS

NoSQL

XML files

JSON files

CSV files

Virtual graphs

Triple Pattern

Fragment Servers
SPARQL endpoint: RDF triplestore (3/4)

- **Standalone server**
- **Windows service**
- **Web application (e.g. in Apache Tomcat, Jetty Boots, etc.)**
- **Cloud server**
- **In-Process (embedded in your own code)**
SPARQL endpoint: simplest configuration (4/4)

- Example 1: [https://dbpedia.org/sparql](https://dbpedia.org/sparql) = public (read only) SPARQL endpoint of DBpedia’s triplestore (Virtuoso)
- Example 2: [https://rdf.ontotext.com/4139541402/mydb/repositories/OpenSmartHomeDataSet](https://rdf.ontotext.com/4139541402/mydb/repositories/OpenSmartHomeDataSet) = public (read only) SPARQL endpoint of Open Smart House (OSH) dataset (GraphDB cloud)
SPARQL queries can be reused everywhere?

- **in theory**: yes (e.g. compliant to SPARQL 1.1)
- **in practice**: watch out
  - < 100% coverage for SPARQL 1.1 standard
  - Different interpretation of some parts of the specification
  - Different behavior towards named graphs and default graph
  - SPARQL extensions (e.g. PATHS, geospatial, temporal, Custom Datatypes (CDT), ...)
  - Reasoning engine
  - Well-formed RDF literals (canonicalization)

- **extensive testing is always needed!**
## SPARQL communication protocol: URLs and options

<table>
<thead>
<tr>
<th>Triplestore</th>
<th>Type</th>
<th>HTTP method</th>
<th>Default local endpoint URL</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache Jena Fuseki</td>
<td>Read</td>
<td>GET/POST</td>
<td><a href="http://localhost:3030/MyDB/query">http://localhost:3030/MyDB/query</a></td>
<td>“query”: containing SPARQL read query</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><a href="http://localhost:3030/MyDB/sparql">http://localhost:3030/MyDB/sparql</a></td>
<td>“Accept”: application/json, etc.</td>
</tr>
<tr>
<td></td>
<td>Update</td>
<td>POST</td>
<td><a href="http://localhost:3030/MyDB/update">http://localhost:3030/MyDB/update</a></td>
<td>“update”: containing SPARQL update query</td>
</tr>
<tr>
<td>Stardog</td>
<td>Read</td>
<td>GET/POST</td>
<td><a href="http://localhost:5820/MyDB/query">http://localhost:5820/MyDB/query</a></td>
<td>“query”: containing SPARQL read query</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>“accept”: application/json, etc.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>“reasoning”: Boolean to activate reasoning</td>
</tr>
<tr>
<td></td>
<td>Update</td>
<td>POST</td>
<td><a href="http://localhost:5820/MyDB/update">http://localhost:5820/MyDB/update</a></td>
<td>“query”/“update”: containing SPARQL update query</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>“reasoning”: Boolean to activate reasoning</td>
</tr>
<tr>
<td>Ontotext GraphDB</td>
<td>Read</td>
<td>GET/POST</td>
<td><a href="http://localhost:7200/repositories/MyDB">http://localhost:7200/repositories/MyDB</a></td>
<td>“query”: containing SPARQL read query</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>“accept”: application/json, etc.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>“infer”: Boolean to activate reasoning</td>
</tr>
<tr>
<td></td>
<td>Update</td>
<td>POST</td>
<td><a href="http://localhost:7200/repositories/MyDB/statements">http://localhost:7200/repositories/MyDB/statements</a></td>
<td>“update”: containing SPARQL update query</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>“infer”: Boolean to activate reasoning</td>
</tr>
<tr>
<td>BBN Parliament</td>
<td>Read</td>
<td>GET/POST</td>
<td><a href="http://localhost:8089/parliament/sparql">http://localhost:8089/parliament/sparql</a></td>
<td>“query”: containing SPARQL read query</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>“Accept”: application/sparql-results+json, etc.</td>
</tr>
<tr>
<td></td>
<td>Update</td>
<td>POST</td>
<td><a href="http://localhost:8089/parliament/sparql">http://localhost:8089/parliament/sparql</a></td>
<td>“update”: containing SPARQL update query</td>
</tr>
</tbody>
</table>
SPARQL communication protocol: media types

- **SPARQL SELECT/ASK results**
  - **JSON**: application/sparql-results+json (sometimes also application/json)
  - **XML**: application/sparql-results+xml (sometimes also application/xml)
  - **TSV**: text/tab-separated-values
  - **CSV**: text/csv
  - **Boolean**: text/Boolean
  - **Binary RDF results**: application/x-binary-rdf-results-table

- **SPARQL CONSTRUCT/DESCRIBE results**
  - **Turtle**: text/turtle (sometimes also application/x-turtle)
  - **JSON-LD**: application/json (sometimes also application/ld+json)
  - **N3**: text/n3
  - **RDF/XML**: application/rdf+xml
  - **TriG**: application/trig
  - **TriX**: application/trix
  - **N-Quads**: application/n-quad
  - **...**
SPARQL communication protocol: response

```
SELECT *
WHERE { ?s ?p ?o }

Accept: application/sparql-results+json
```