Comparison of graph databases and triplestores on the example of the JVMG Project

Tobias Malmsheimer Stuttgart Media University
Japanese Visual Media Graph

- Research project at Stuttgart Media University
- Goal: Build a knowledge graph on the domain of Japanese visual media (manga, anime, computer games)
- Data source: Enthusiast communities on the web

- Ingestion workflow
  - Retrieve data dump / load data from API in proprietary data format
  - Assess data structure and organisation and create an OWL ontology
  - Convert into RDF triples
  - Load into Fuseki database

- Integration workflow
  - Match semantically equivalent entities into clusters
  - Merge information from all sources (create a merged entity)
  - Match semantically equivalent entity properties
  - Merge property values (reduce redundancy)
Why we need a Database

- Need to store and access our knowledge graph
  - ttl's are boring

- Our frontend expects a sparql-endpoint
  - User can explore our knowledge graph

- Tiny use cases (TUCs)
  - Media researcher use the knowledge graph for their research questions
  - In this way, we can improve the knowledge graph and the supporting tools
How did I test

- **Subjective**
  - Ease of use
    - Get the database and install it
    - Loading data
    - Interface and help

- **Objective**
  - Performance
    - CPU: time for sparql requests
    - RAM usage: after start and while running requests
    - Disk usage: after loading data

- **Subset of our knowledge graph**
  - vndb ~4.8 mio triples
How does vndb look like

Source: https://zenodo.org/record/5506936
How does vndb look like

<table>
<thead>
<tr>
<th>Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Character</td>
<td>90077</td>
</tr>
<tr>
<td>Trait</td>
<td>2237</td>
</tr>
<tr>
<td>Tag</td>
<td>2053</td>
</tr>
<tr>
<td>Producer</td>
<td>10394</td>
</tr>
<tr>
<td>Staff</td>
<td>21164</td>
</tr>
<tr>
<td>Visual Novel</td>
<td>28190</td>
</tr>
<tr>
<td>Visual Novel Release</td>
<td>71349</td>
</tr>
</tbody>
</table>
Disclaimer

- Just my opinion
- I did not optimize everything for each database
- Interested in the “out of the box” experience
- Therefore critique and bad numbers do not mean the database is bad
## Some Facts

<table>
<thead>
<tr>
<th>Datenbank</th>
<th>Creator</th>
<th>License</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fuseki</td>
<td>Apache Foundation</td>
<td>Apache Version 2.0</td>
</tr>
<tr>
<td>Virtuoso - Community</td>
<td>OpenLink Software</td>
<td>GPL</td>
</tr>
<tr>
<td>GraphDB - Free Version</td>
<td>Ontotext Sirma Group</td>
<td>proprietary</td>
</tr>
<tr>
<td>Blazegraph</td>
<td>Systap</td>
<td>GPL-2.0</td>
</tr>
</tbody>
</table>
## Subjective Comparison

<table>
<thead>
<tr>
<th>Database</th>
<th>Download and Start</th>
<th>Gui</th>
<th>Loading Data</th>
<th>Help Documentation</th>
<th>Overall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fuseki</td>
<td>Just download and start</td>
<td>Manage Queries, Datasets</td>
<td>Web gui or terminal</td>
<td>Just some links</td>
<td>Not fancy, gets the job done</td>
</tr>
<tr>
<td>Virtuoso - Community</td>
<td>Download, built and start</td>
<td>Many options, but convoluted</td>
<td>Web gui or terminal</td>
<td>Good documentation</td>
<td>unresponsive gui, some clicks timeout</td>
</tr>
<tr>
<td>GraphDB - Free Version</td>
<td>Registration needed</td>
<td>Manage Queries and Repositories</td>
<td>Web gui or terminal, complains about triple</td>
<td>Nice Web Help</td>
<td>Strange RAM limit</td>
</tr>
<tr>
<td>Blazegraph</td>
<td>Just download and start</td>
<td>Basic: manage queries</td>
<td>Web gui</td>
<td>Some dead links</td>
<td>Strange and slow in comparison</td>
</tr>
</tbody>
</table>
Performance - Queries

- 3 different queries
  - Usual Queries we often use
- Small Query
  - Character page from Phorni
  - 96 triple
  - Small character page
- Medium Query
  - Overview of all vndb Characters
  - ~180k triple
  - Name of every entity which has type Character
- Big Query
  - Tuc 1
  - ~1.7 mio triple
  - Which trait and vn-tag correlates with other traits and vn-tags on character basis
Performance - CPU

- Python
  - Sparqlwrapper and rdflib for sparql endpoints

- Complete round trip
  - Executing sparql query
  - Serializing result

- Two kinds of queries
  - Select, assumption: better optimized on most dbs
    - Not used for TUC 1
  - Construct, to get subgraphs
    - Used in all queries

- First and cached queries
  - Every query gets executed 6 times
  - Time for the first query (when data is still on disk)
  - Average of the 5 next queries to measure potential caching speedups
Small Query

Database

<table>
<thead>
<tr>
<th>Database</th>
<th>select - first</th>
<th>select - cached</th>
<th>construct - first</th>
<th>construct - cached</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fuseki</td>
<td>0.09</td>
<td>0.01</td>
<td>0.13</td>
<td>0.03</td>
</tr>
<tr>
<td>Blazegraph</td>
<td>1.65</td>
<td>1.57</td>
<td>1.73</td>
<td>3.27</td>
</tr>
<tr>
<td>Virtuoso</td>
<td>0.08</td>
<td>0.002</td>
<td>0.07</td>
<td>0.007</td>
</tr>
<tr>
<td>GraphDB</td>
<td>0.11</td>
<td>0.01</td>
<td>0.1</td>
<td>0.02</td>
</tr>
</tbody>
</table>
Medium Query

![Bar chart showing performance metrics for different databases]

- **Fuseki**:
  - select - first: 2.94
  - select - cached: 2.34
  - construct - first: 9.65
  - construct - cached: 10.71

- **Blazegraph**:
  - select - first: 10.26
  - select - cached: 10.83
  - construct - first: 12.51
  - construct - cached: 13.7

- **Virtuoso**:
  - select - first: 0.7
  - select - cached: 0.72
  - construct - first: 13.3
  - construct - cached: 13.8

- **GraphDB**:
  - select - first: 1.6
  - select - cached: 1.35
  - construct - first: 11.66
  - construct - cached: 14.07
Big Query

- Fuseki: 15 min
- Blazegraph: crash - out of heap (> 24gb)
- Virtuoso: 1.5 h
- GraphDB: 19 min
Ram usage

- pmap
  - Virtual memory
  - Resident
  - Dirty
- Measured
  - After database start
  - While big query
## RAM usage

<table>
<thead>
<tr>
<th>DB</th>
<th>Virtual</th>
<th>Resident</th>
<th>Dirty</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fuseki</td>
<td>23 GB</td>
<td>0.23 GB</td>
<td>0.2 GB</td>
</tr>
<tr>
<td>Blazegraph</td>
<td>25 GB</td>
<td>0.16 GB</td>
<td>0.14 GB</td>
</tr>
<tr>
<td>Virtuoso</td>
<td>1.9 GB</td>
<td>0.47 GB</td>
<td>0.45 GB</td>
</tr>
<tr>
<td>GraphDB</td>
<td>15 GB</td>
<td>1.6 GB</td>
<td>1.58 GB</td>
</tr>
<tr>
<td>Fuseki</td>
<td>25 GB</td>
<td>2.1 GB</td>
<td>1.8 GB</td>
</tr>
<tr>
<td>Blazegraph*</td>
<td>39 GB</td>
<td>27 GB</td>
<td>27 GB</td>
</tr>
<tr>
<td>Virtuoso</td>
<td>13 GB</td>
<td>10 GB</td>
<td>9 GB</td>
</tr>
<tr>
<td>GraphDB</td>
<td>15 GB</td>
<td>1.8 GB</td>
<td>1.8 GB</td>
</tr>
</tbody>
</table>
Disk usage

- Just the data when loaded into the database
- Not the whole database
- Measured with: du -h

<table>
<thead>
<tr>
<th>DB</th>
<th>Disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>ttl-file</td>
<td>200 MB</td>
</tr>
<tr>
<td>Fuseki</td>
<td>1.6 GB</td>
</tr>
<tr>
<td>Blazegraph</td>
<td>780 MB</td>
</tr>
<tr>
<td>Virtuoso</td>
<td>605 MB</td>
</tr>
<tr>
<td>GraphDB</td>
<td>541 MB</td>
</tr>
</tbody>
</table>
Lessons learned

- Taking a look at other databases is quite interesting
  - Select vs Construct Performance
- Other type of databases where interesting but not useful
  - Enjoyed neo4j a lot, but a property graph model and a rdf knowledge graph are too different
  - Also showed us how important a good tutorial/guide is
- Still unsure what we did wrong with Blazegraph
- Fuseki: dead weight does not matter, the same queries are not slower with all our triples loaded
- Some have very weird licenses, like Allegrograph, which does not allow you publish your findings without their consent
Questions?