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)

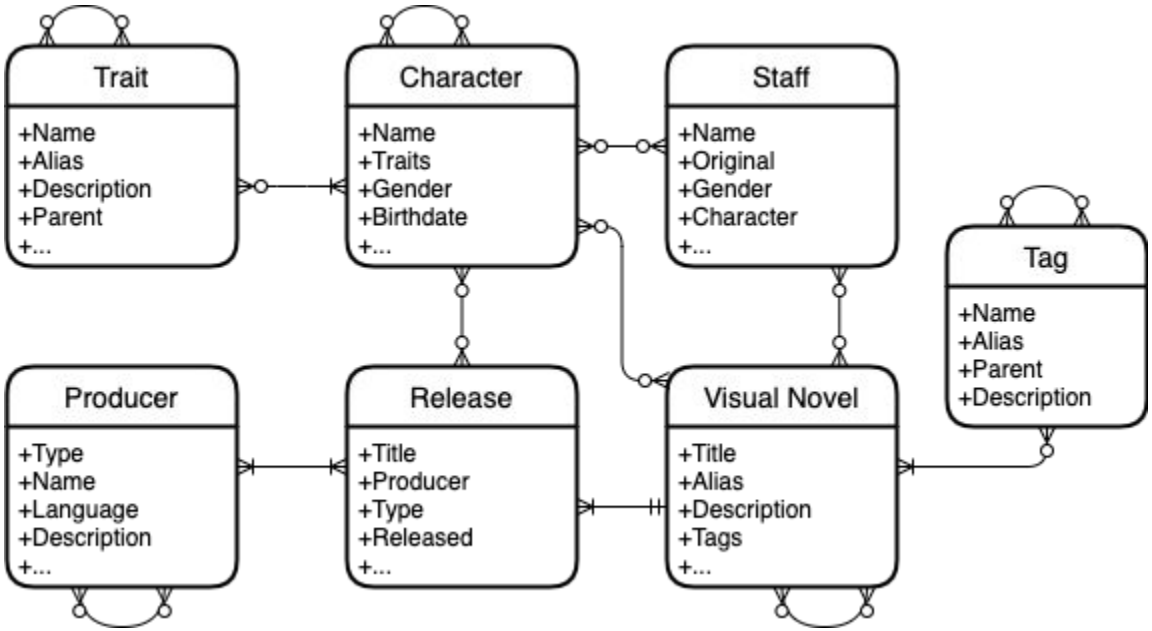
Why we need a Database

- Need to store and access our knowledge graph
  - ttls 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

Type	Count
Character	90077
Trait	2237
Tag	2053
Producer	10394
Staff	21164
Visual Novel	28190
Visual Novel Release	71349

Disclaimer

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

Some Facts

Datenbank	Creator	License
Fuseki	Apache Foundation	Apache Version 2.0
Virtuoso - Community	OpenLink Software	GPL
GraphDB - Free Version	Ontotext Sirma Group	proprietary
Blazegraph	Systap	GPL-2.0

Subjective Comparison

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

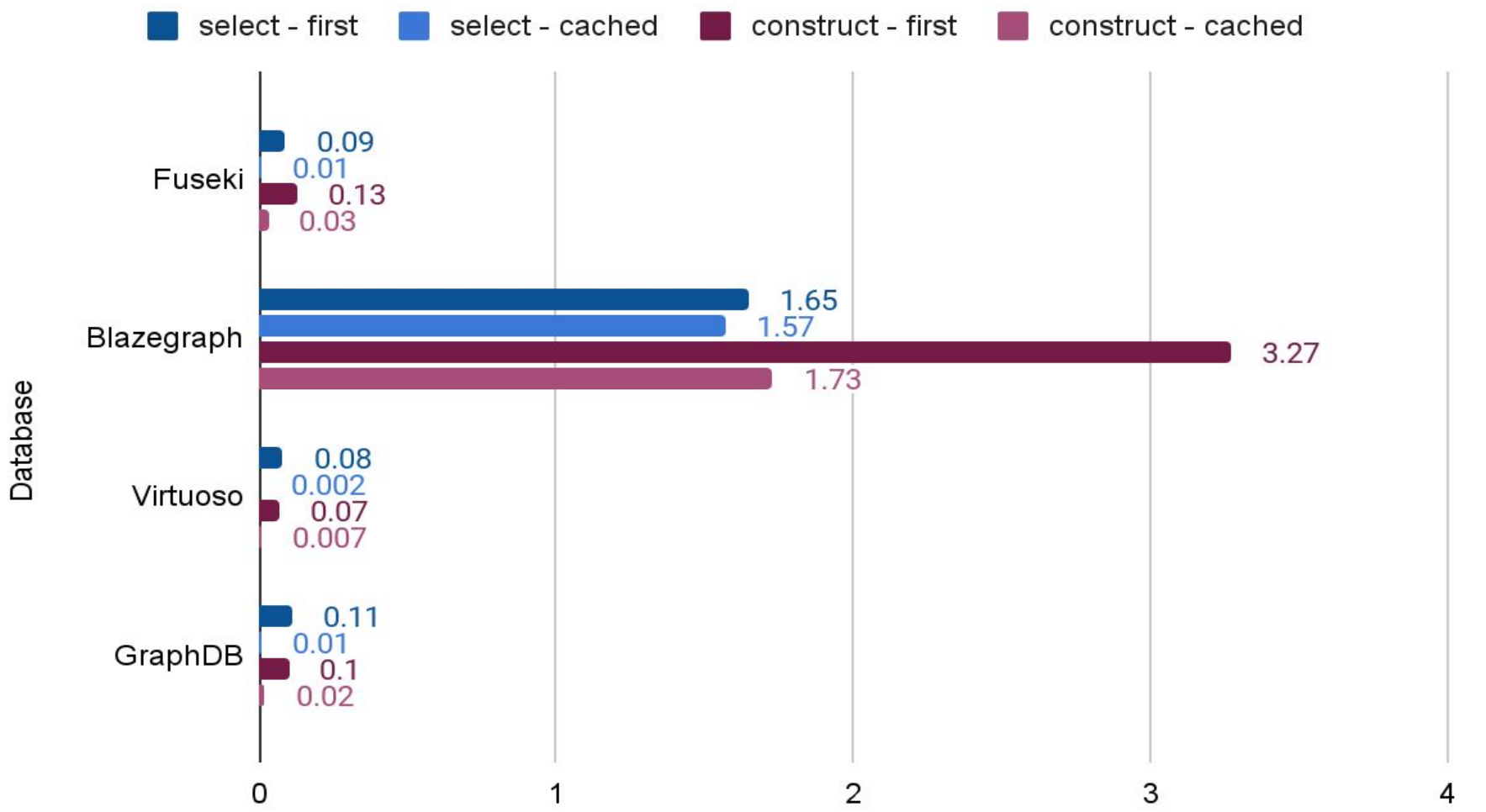
Performance - Queries

- 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

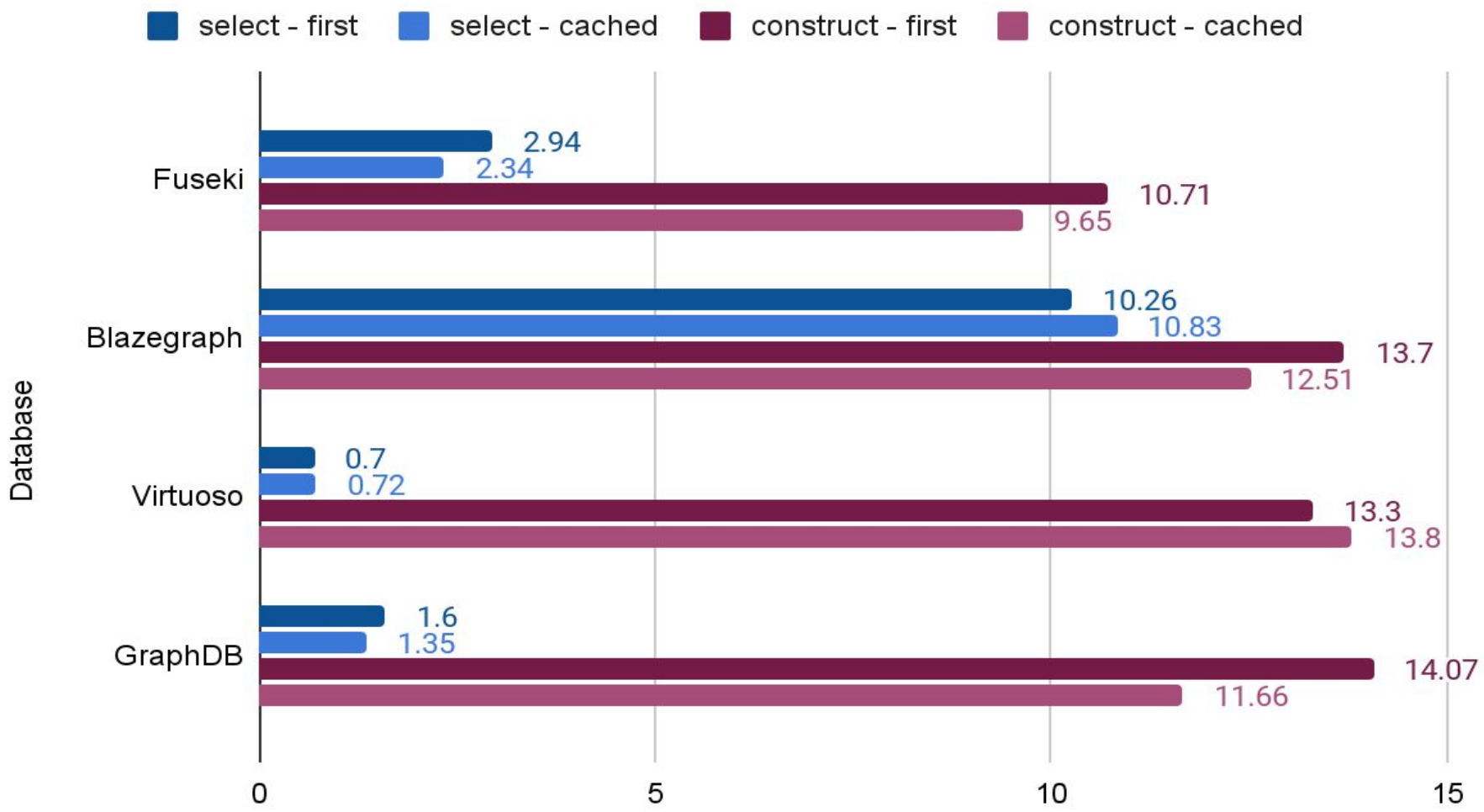
Performance - CPU

- 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


select - first
select - cached
construct - first
construct - cached
Database
Fuseki
0.09
0.01
0.13
0.03
Blazegraph
1.65
1.57
3.27
1.73
Virtuoso
0.08
0.002
0.07
0.007
GraphDB
0.11
0.01
0.1
0.02
0
1
2
3
4

Medium Query



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

DB	Virtual	Resident	Dirty
Fuseki	23 GB	0.23 GB	0.2 GB
Blazegraph	25 GB	0.16 GB	0.14 GB
Virtuoso	1.9 GB	0.47 GB	0.45 GB
GraphDB	15 GB	1.6 GB	1.58 GB
Fuseki	25 GB	2.1 GB	1.8 GB
Blazegraph*	39 GB	27 GB	27 GB
Virtuoso	13 GB	10 GB	9 GB
GraphDB	15 GB	1.8 GB	1.8 GB

Disk usage

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

DB	**Disk**
ttl-file	200 MB
Fuseki	1.6 GB
Blazegraph	780 MB
Virtuoso	605 MB
GraphDB	541 MB

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?