PADERBORN UNIVERSITY

Work Package 3

Storage Layer

Alexander Bigerl

RAKI Ablschlusstreffen – Paderborn

December 14, 2022

RAKI

Förderkennzeichen: 01MD19012B

Gefördert durch:

Bundesministerium für Wirtschaft und Klimaschutz

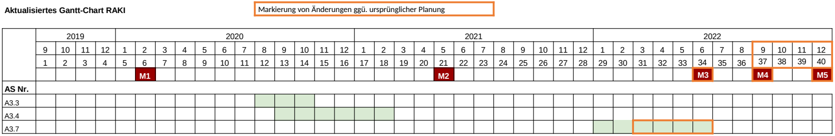
aufgrund eines Beschlusses des Deutschen Bundestages

Introduction

TENTRIS

Contents

Work Packages



- WP 3.3: Evaluation of the storage layer
- WP 3.4: Implementation of optimizations
- WP 3.7: Final performance evaluation

Motivation

SPARQL Backend for Reasoning

Instance checking is a bottleneck for CEL on large datasets.

- Existing closed-world reasoners do not scale.
- Previous findings [1] suggest that SPARQL-engines scale much better.

WP 3.3: Identify Most Promising System

Stresstest

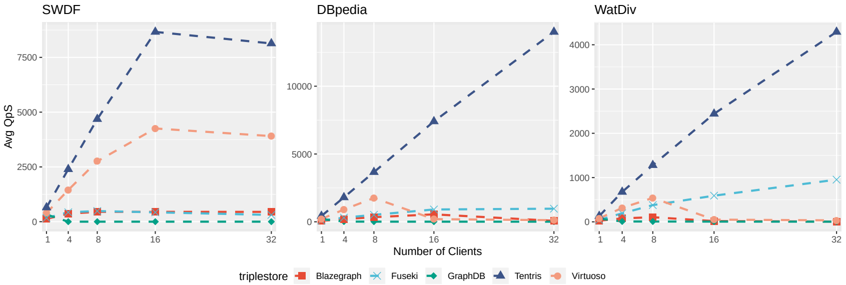


Figure: The plots report average queries per second (Avg QpS) for different triple stores. The subplots report results for benchmarks based on different datasets with increasing size.

→ TENTRIS

Opportunities for Improvements

Tentris

- Improve Storage efficiency
- Add missing SPARQL query features

Optimizing Storage Efficiency

Optimizing Storage Efficiency

Hashing the Hypertrie

Subject

- HYPERTRIE, a sparse tensor data structure used in the triple store TENTRIS[1]

Goal

- Reduce storage footprint

Approach

- Eliminating redundancies
- Reducing storage overhead

Results

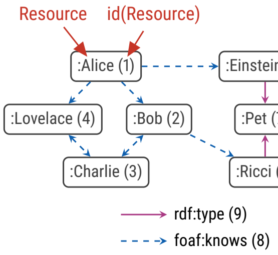
- Up to 70% improved storage footprint
- Up to 7x faster execution of query mixes

#hypertrie

[1] Bigerl et al. (2020) TENTRIS – A Tensor-Based Triple Store. In: ISWC 2020

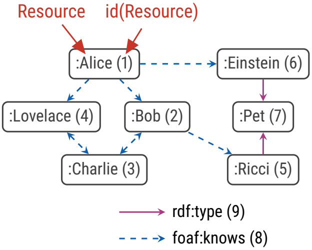
RDF to Tensor

Mapping



RDF to Tensor

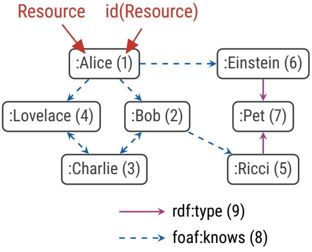
Mapping



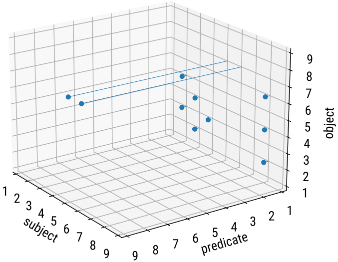
id(s)	*id*(p)	*id*(o)
8	1	2
8	1	4
8	1	6
2	1	3
2	1	5
3	1	2
3	1	4
4	1	3
5	9	7
6	9	7

RDF to Tensor

Mapping



id(s)	*id*(p)	*id*(o)
8	1	2
8	1	4
8	1	6
2	1	3
2	1	5
3	1	2
3	1	4
4	1	3
5	9	7
6	9	7



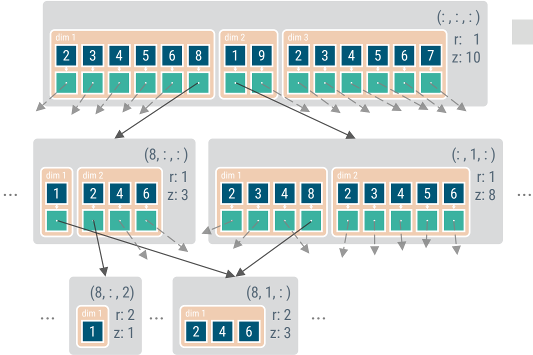
Baseline Hypertrie

Design

Requirements

1. Allow efficient slicing …
2. …by any dimensions
3. Iterate non-zero slices of any dimension
4. Memory-efficient

id(s)	id(p)	id(o)
8	1	2
8	1	4
8	1	6
2	1	3
2	1	5
3	1	2
3	1	4
4	1	3
5	9	7
6	9	7



Baseline Hypertrie

Eliminating Structural Redundancies

Naive storage bound

$$\mathcal{O}(d! \cdot d \cdot z)$$

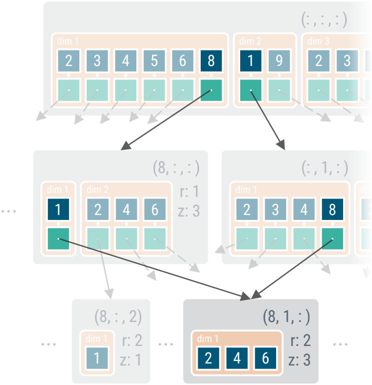
Equal slices stored only once

$$\mathcal{O}(2^{d-1} \cdot d \cdot z)$$

Upper bound for storing triples

all tries: $6 \cdot z$

hypertrie: $4 \cdot z$



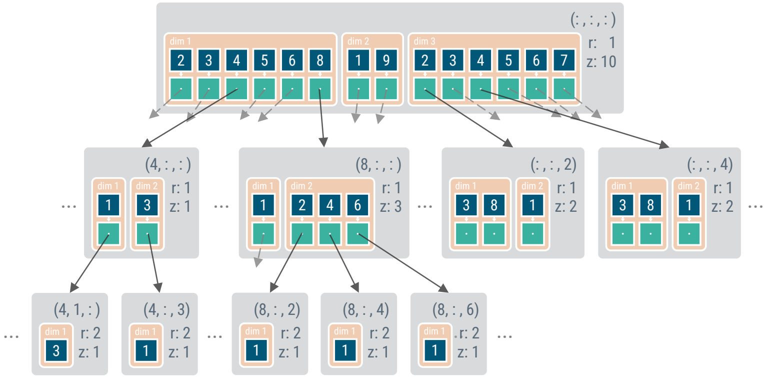
---

d: tensor dimensions

z: stored entries

Optimizations

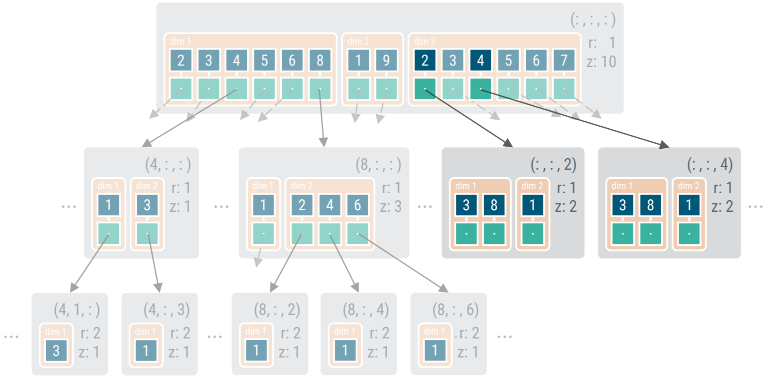
Deduplicate Nodes



(x, y, z): slice key to root node
r: reference count
z: stored entries

Optimizations

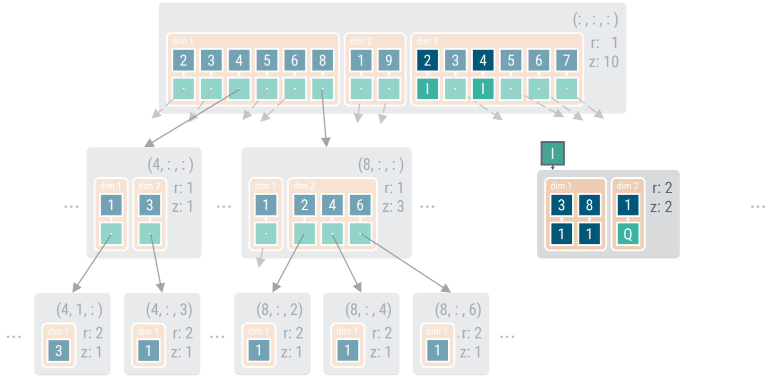
Deduplicate Nodes



(x, y, z): slice key to root node
r: reference count
z: stored entries

Optimizations

Deduplicate Nodes



(x, y, z): slice key to root node
r: reference count
z: stored entries

Optimizations

Hashing-Based Identifiers

Let j be an order-dependent hashing scheme for HYPERTRIE entries.
Hash of a HYPERTRIE h:

$$i(h) := \bigoplus_{\mathbf{k} \in \mathrm{dom}(h)} j(\mathbf{k}) \tag{1}$$

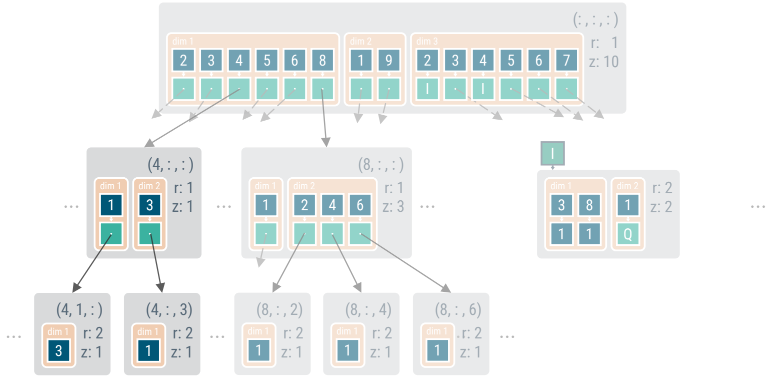
Update hash after adding/removing entry $\mathbf{k}'$ in $\mathcal{O}(1)$:

$$i(h) \oplus j(\mathbf{k}') \tag{2}$$

dom: set of non-zero entries of a HYPERTRIE

Optimizations

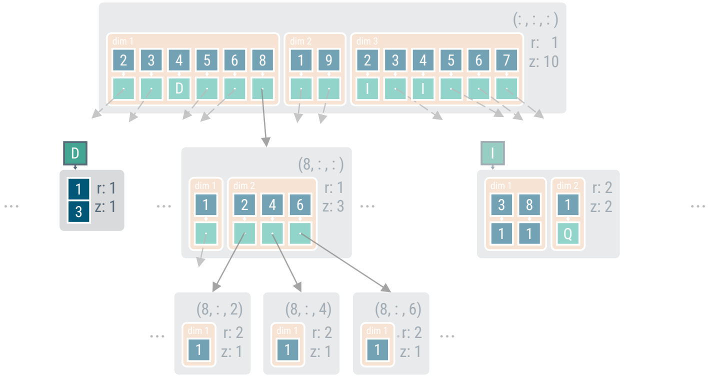
Compact Single Entry Nodes



(x, y, z): slice key to root node
r: reference count
z: stored entries

Optimizations

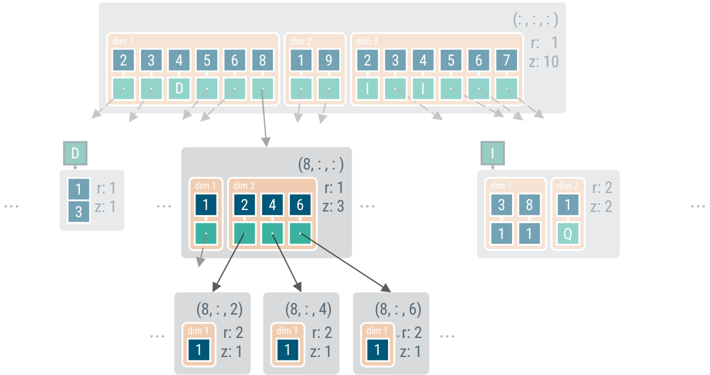
Compact Single Entry Nodes



(x, y, z): slice key to root node
r: reference count
z: stored entries

Optimizations

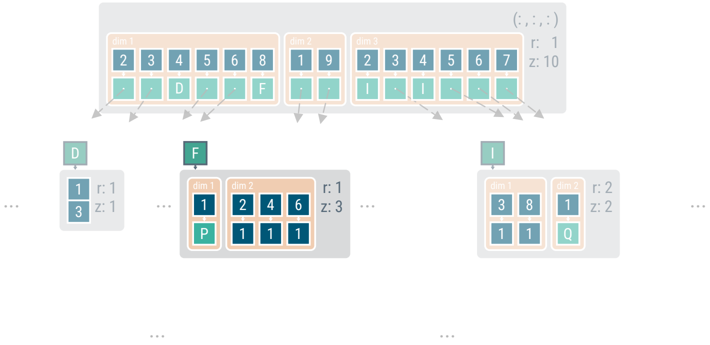
Eliminate Single Entry Leaf Nodes



(x, y, z): slice key to root node
r: reference count
z: stored entries

Optimizations

Eliminate Single Entry Leaf Nodes

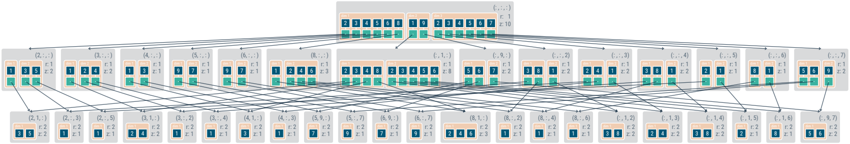


(x, y, z): slice key to root node
r: reference count
z: stored entries

Optimizations

Complete Hypertrie

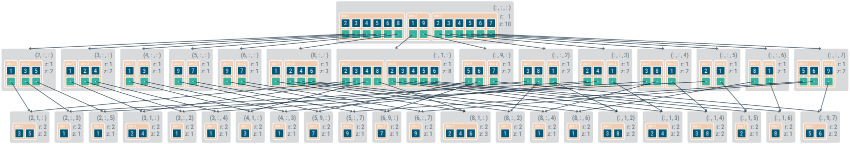
Baseline Hypertrie



Optimizations

Complete Hypertrie

Baseline Hypertrie



Optimized Hypertrie

Evaluation

Setup

Setup

- IGUANA-based stress tests
- 30 runs $\times$ {SWDF, DBpedia, WatDiv, Wikidata} benchmarks

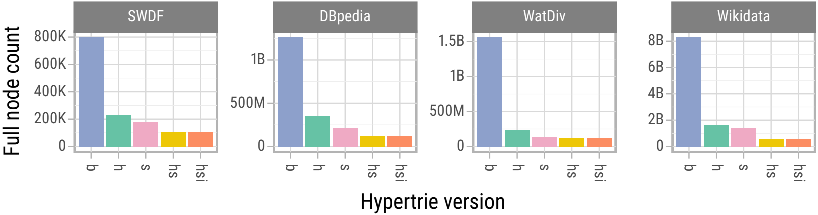
Queries

- SELECT, opt. DISTINCT & Basic Graph Pattern
- Generated SWDF, DBpedia & Wikidata with FEASIBLE
- Basic templates for WatDiv & opt. DISTINCT

Dataset	Type	#T	#Q
SWDF	real-world	372 k	203
DBpedia	real-world	681 M	554
WatDiv	synthetic	1 G	45
Wikidata	real-world	5.5 G	495

Evaluation

Full Node Counts

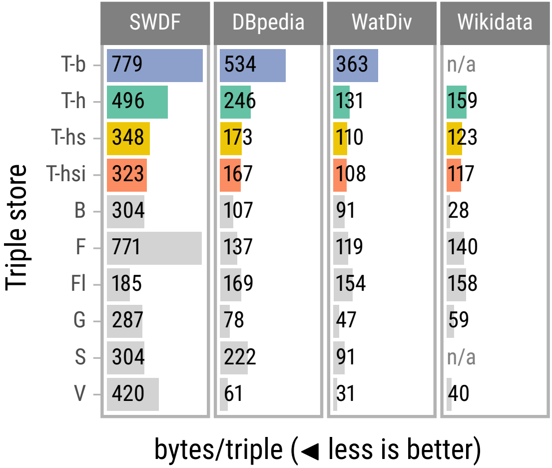


Hypertrie versions

- **b** Baseline
- **s** Single-entry node
- **h** Hash identifiers
- **i** In-place storage of single-entry leaf nodes

Evaluation

Storage Efficiency



Hypertrie versions

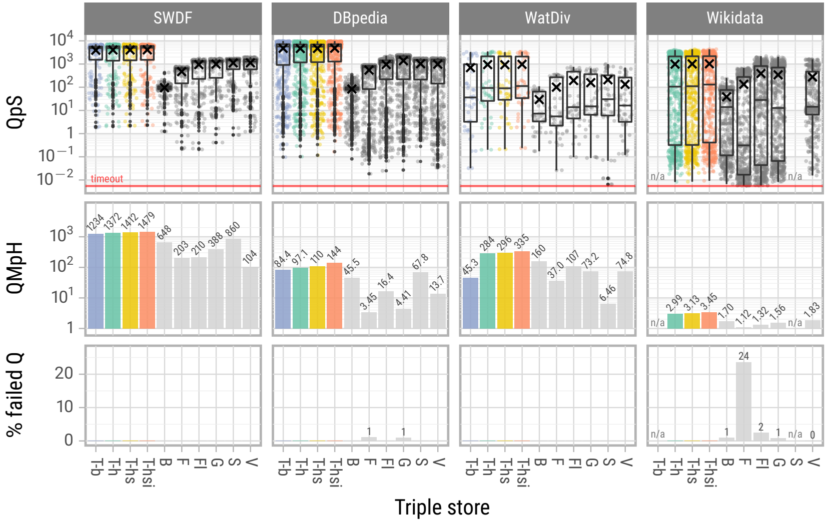
b	Baseline
s	Single-entry node
h	Hash identifiers
i	In-place storage of single-entry leaf nodes

Triple stores

T	Tentris
B	Blazegraph
F	Fuseki
Fl	Fuseki LTJ
G	GraphDB
S	gStore
V	Virtuoso

Evaluation

Stresstest



Hashing the HYPERTRIE

Summary

- Reduced storage footprint
- Stable loading performance
- Improved query answer time

#hypertrie

Missing SPARQL features

Missing SPARQL features

OWL to SPARQL

ALC Class Expression C_i	SPARQL Graph Pattern $\tau(C_i, ?var)$
A	$\{ ?var$ `rdf:type` $A\ .\ \}$
$\neg C$	$\{ ?var\ ?p\ ?o\ .$ `FILTER NOT EXISTS` $\{ \tau(C, ?var) \} \}$
$C_1 \sqcap \cdots \sqcap C_n$	$\{ \tau(C_1, ?var) \ldots \tau(C_n, ?var) \}$
$C_1 \sqcup \cdots \sqcup C_n$	$\{ \{ \tau(C_1, ?var) \}$ `UNION` ... `UNION` $\{ \tau(C_n, ?var) \} \}$
$\exists r.C$	$\{ ?var\ r\ ?s\ .\} \tau(C_i, ?s)$
$\forall r.C$	$\{ ?var\ r\ ?s0\ .\ \{$ `SELECT` $?var$ `(COUNT(?s1) AS ?c1) WHERE` $\{ ?var\ r\ ?s1\ .\ \tau(C, ?vs1) \}$ `GROUP BY` $?var \}$ $\{$ `SELECT` $?var$ `(COUNT(?s2) AS ?c2) WHERE` $\{ ?var\ r\ ?s2 \}$ `GROUP BY` $?var \}$ `FILTER`$(?c1 = ?c2) \}$

Table: The mapping of ALC expressions to SPARQL queries [1].

Missing SPARQL features

OWL to SPARQL

TENTRIS was extended by the features:

- Union
- Filters and negative pattern matching
- Simple subqueries
- Grouping and aggregates

First multi-way join SPARQL implementation with extended features.

WP 3.7: Final Evaluation

Setup

Setup

- IGUANA-based stress tests
- 4 benchmarking datasets

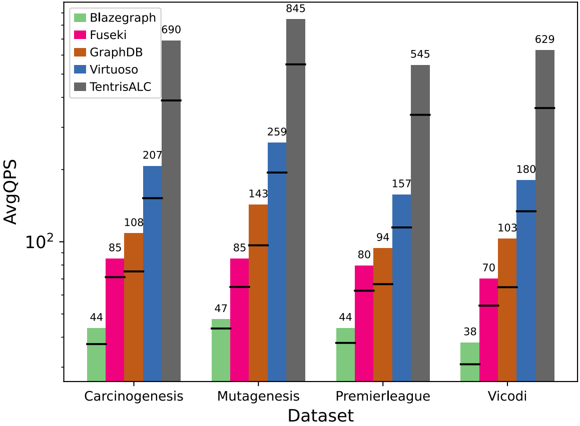
Queries

- Class expressions generated with OntoLearn
- Generated 300 random ALC class expressions per dataset
- Class expressions translated to SPARQL

Dataset	#T	#S	#P	#O
Carcinogenesis	157K	22.5K	25	23.2K
Mutagenesis	96K	14.2K	16	15K
Premier League	2.1M	11.5K	217	12.5K
Vicodi	405K	33.4K	14	35.2K

WP 3.7: Final Evaluation

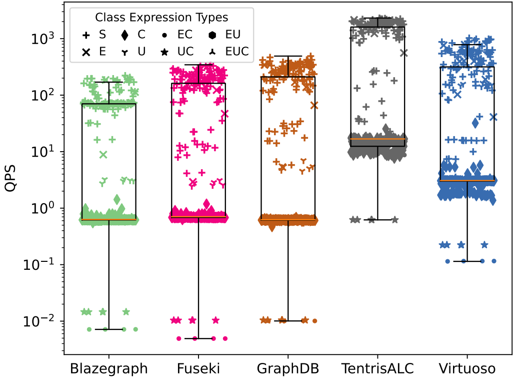
Benchmarking Results



SPARQL-based Instance retrieval benchmarking results reporing average queries per second (AvgQPS).

WP 3.7: Final Evaluation

Benchmarking Results (Detail)



Vicodi dataset

Legend:

- S simple
- C negations
- U universal quantifier
- E existential restriction

Scientific Publications

Published works

- Tentris – A Tensor-based Triple Store, ISWC 2020 [2]
- Hashing the Hypertrie: Space- and Time-Efficient Indexing for SPARQL in Tensors, ISWC 2022 [3]

Upcomming works

- Rapid Execution of GraphQL Queries over Tensors, VLDB 2023 (under review)
- A Recursive Evaluation of SPARQL Queries Accelerates Class Expression Learning, ESWC 2023 (under submission)

References I

[1] S. Bin, L. Bühmann, J. Lehmann, and A.-C. Ngonga Ngomo, "Towards sparql-based induction for large-scale rdf data sets," in *ECAI 2016*, pp. 1551–1552, IOS Press, 2016.

[2] A. Bigerl, F. Conrads, C. Behning, M. A. Sherif, M. Saleem, and A.-C. Ngonga Ngomo, "Tentris – A Tensor-Based Triple Store," in *The Semantic Web – ISWC 2020*, pp. 56–73, Springer International Publishing, 2020.

[3] A. Bigerl, L. Conrads, C. Behning, M. A. Sherif, M. Saleem, and A.-C. Ngonga Ngomo, "Hashing the Hypertrie: Space- and Time-Efficient Indexing for SPARQL in Tensors," in *The Semantic Web – ISWC 2022*, Springer International Publishing, 2022.

Acknowledgement

Förderkennzeichen: 01MD19012B

Gefördert durch:

Bundesministerium
für Wirtschaft
und Klimaschutz

aufgrund eines Beschlusses
des Deutschen Bundestages