RDF Engines

Stefan Schuh

December 5, 2008

Resource Description Framework

RDF is an acronym for Resource Description Framework

- RDF is a data model to represent meta data.
- RDF was invented an pulished by the W3C in '98.
- RDF was revisited in 2004.

further informations http://www.w3.org/RDF/

Nowadays there is a lot of RDF Data in the world wide web.
For instance:

- Wikipedia
- Sience Life
- Social Networks

Resource Description Framework

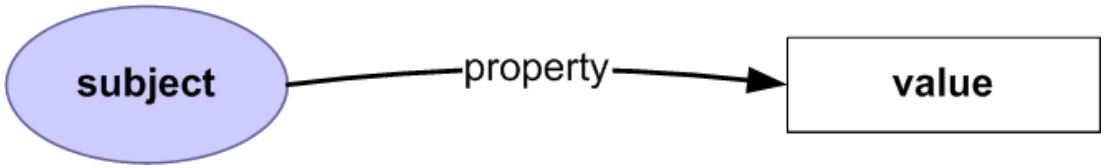
But why is the RDF data model that important?

- RDF allows to store and describe meta data in a standardized way.
- The simplicity allows to describe all kinds of data.
- Important for Semantic Web and Web 2.0

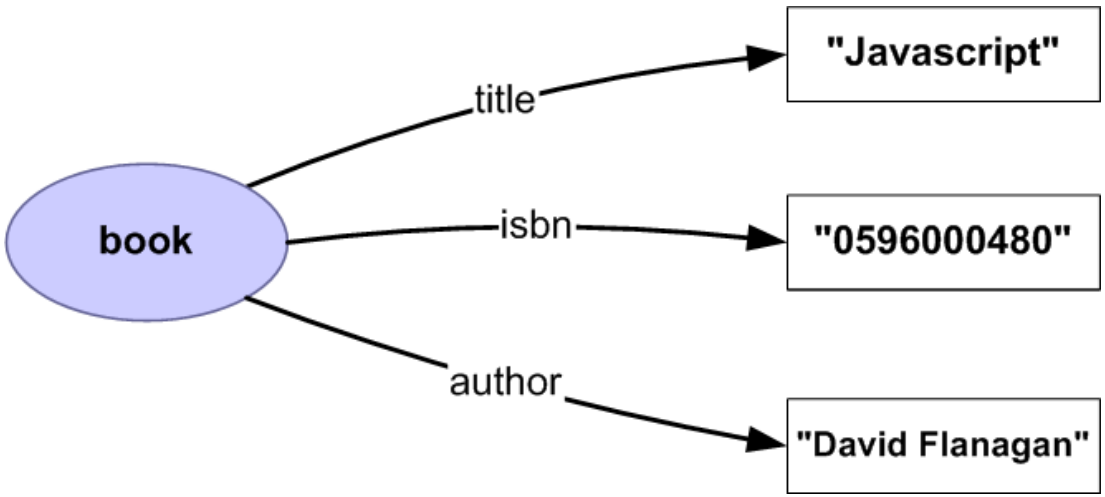
Structuring RDF

RDF structure

- In RDF data is represented as triples.
- The structure of such a triple is (subject,property,value).



Example



But how to query data from such a RDF ??

Query Language SPARQL

What is SPARQL ?

- SPARQL is a standart language to query RDF data.
- SPARQL was recommended in the year 2008 by W3C.

How does a SPARQL query look like ??

How does a SPARQL query look like ?

A simple generic SPARQL query

```
Select ?var1 ?var2 ....
    where { pattern1.  pattern2.  .....}
```

SPARQL Patterns

Triple Patterns	
p1	(s,p,o)
p2	(?s,p,o)
p3	(s,?p,o)
p4	(s,p,?o)
p5	(?s,?p,o)
p6	(s,?p,?o)
p7	(?s,p,?o)
p8	(?s,?p,?o)

Variables are characterized with questionmarks.

Variables are bound such that the triple is in the data set.

A simple query:

select ?i

where { (book,isbn,?i). }

This query yields the isbn number of the book 'book' in our graph example.

Thus the result is the number 0596000480.

Joins:

select ...

where { (?a,b,c).(?a,e,f). ... }

Triple Storages

RDF Graph is represented by set of all triples (s,p,o). Often the triples are stored in a relational Database.

The Problem: How to store these triples in such a way that queries can be answered fast?

Some approaches

Four different approaches

1. Giant Triple Storage
2. Property Tables
3. Vertically Partition Tables
4. Hexastore

Giant Triple storage

Basic idea

- Just store all triples in a single table.
- Important for the performance is the choice of the right indeces, clustered and unclustered.

LUBM like example

LUBM Example:

Subj	Property	Obj
ID1	type	FullProfessor
ID1	teacherOf	'AI'
ID1	bachelorFrom	'MIT'
ID1	mastersFrom	'Cambridge'
ID1	phdFrom	'Yale'
ID2	type	AssocProfessor
ID2	worksFor	'MIT'
ID2	teacherOf	'DataBases'
ID2	bachelorsFrom	'Yale'
ID2	phdFrom	'Stanford'
ID3	type	GradStudent
ID3	advisor	ID2
ID3	teachingAssist	'AI'
ID3	bachelorsFrom	'Stanford'
ID3	mastersFrom	'Princeton'
ID4	type	GradStudent
ID4	advisor	ID1
ID4	takesCourse	'DataBases'
ID4	bachelorsFrom	'Columbia'

Pros and Cons

Pros

- Easy to implement
- It's suitable for a large number of poperties
  - If the right indeces has been choosen

Cons

- Many self joins needed

Some approaches

Four different approaches

1. Giant Triple Storage
2. Property Tables
3. Vertically Partition Tables
4. Hexastore

What is the idea?

- Get relational Schemata out of the RDF data

How can this be achieved?

- Collect several properties in one table

A basic example

Professors

ID	type	teacherOf	bachelorsFrom	mastersFrom	phdFrom	worksFor
ID1	FullProfessor	'AI'	'MIT'	'Cambrige'	'Yale'	NULL
ID2	AssocProfessor	'DataBases'	'Yale'	NULL	'Stanford'	'MIT'

Students

ID	type	advisor	bachelorsFrom	mastersFrom	teachingAssist	takesCourse
ID3	GradStudent	ID2	'Stanford'	'Princeton'	'AI'	NULL
ID4	GradStudent	ID1	'Columbia'	NULL	NULL	'DataBases'

Pros and Cons

Pros

- If the data is structured then we will have nice relational tables.

Cons

- imposes structure on semistructured data
- this yields NULL entries
- the selection of properties to store in one table is non trivial

Some approaches

Four different approaches

1. Giant Triple Storage
2. Property Tables
3. Vertically Partition Tables
4. Hexastore

Vertical Partitioned Table

The basic idea of the vertical partinioned table approache:

- Generate for every distinct property an own two column table.

type

ID1	FullProfessor
ID2	AssocProfessor
ID3	GradStudent
ID4	GradStudent

teacherOf

ID1	'AI'
ID2	'DataBases'

bachelorsFrom

ID1	'MIT'
ID2	'Yale'
ID3	'Stanford'
ID4	'Columbia'

mastersFrom

ID1	'Cambridge'
ID3	'Princeton'

phdFrom

ID1	'Yale'
ID2	'Stanford'

worksFor

ID2	'MIT'

advisor

ID3	ID2
ID4	ID1

bachelorsFrom

ID1	'MIT'
ID2	'Yale'
ID3	'Stanford'
ID4	'Columbia'

teachingAssist

ID3	'AI'

takesCourse

ID4	'DataBases'

Pros and Cons

Pros

- Excellent performance
  - Small number of properties
  - Queries are bound on just a few properties

Cons

- Bad performance
  - When a property of a query is not bound
  - When a property is bound in the running time
  - When there exists a huge number of properties

Some approaches

Four different approaches

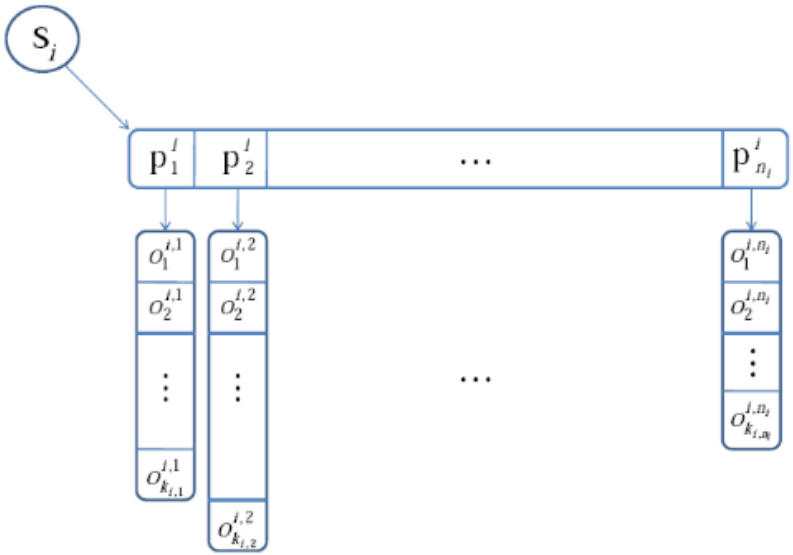
1. Giant Triple Storage
2. Property Tables
3. Vertically Partitioned Tables
4. Hexastore

The basic idea of Hexastore

- An index for each permutaion is materialized. In the Hexastore solution indeces for all orderings are generated: Namely **spo**,**pos**,**osp**,**sop**,**pso**,**ops**
- This index should give you a fast response time.

The hexastore index structure

How does such an index look like in the hexastore?



An example

The **spo** index :

Subject key S_i points to sorted vector of n_i property keys, $\{p_1^i, p_2^i, ..., p_{n_i}^i\}$.
Each property key p_j^i is linked to a sorted list of object keys.

The object key lists are shared with the **pso** index.

Pros and Cons

Pros

- Fast Merge Joins in the first steps of any query.

Cons

- Up to five fold memory usage.
- Optimized for memory based storage.
  - might have bad performance on disk based storage.

Experimental Evaluation - Motivation

We have seen four approaches,
but wich is the best one?

Experimental Evaluation - Introduction

How are we going to test these approaches?

1. We will build some reference implementation of the different aproaches using state of the art techniques.
2. The reference implementations will be testet with real data sets or synthetic data sets.
   - LUBM - synthetic data set
   - Barton - real data set
3. We will query a small set of querys to the data sets.
4. Benchmark the response time.

First Experiment - Triple Store vs Vertically Partitioned

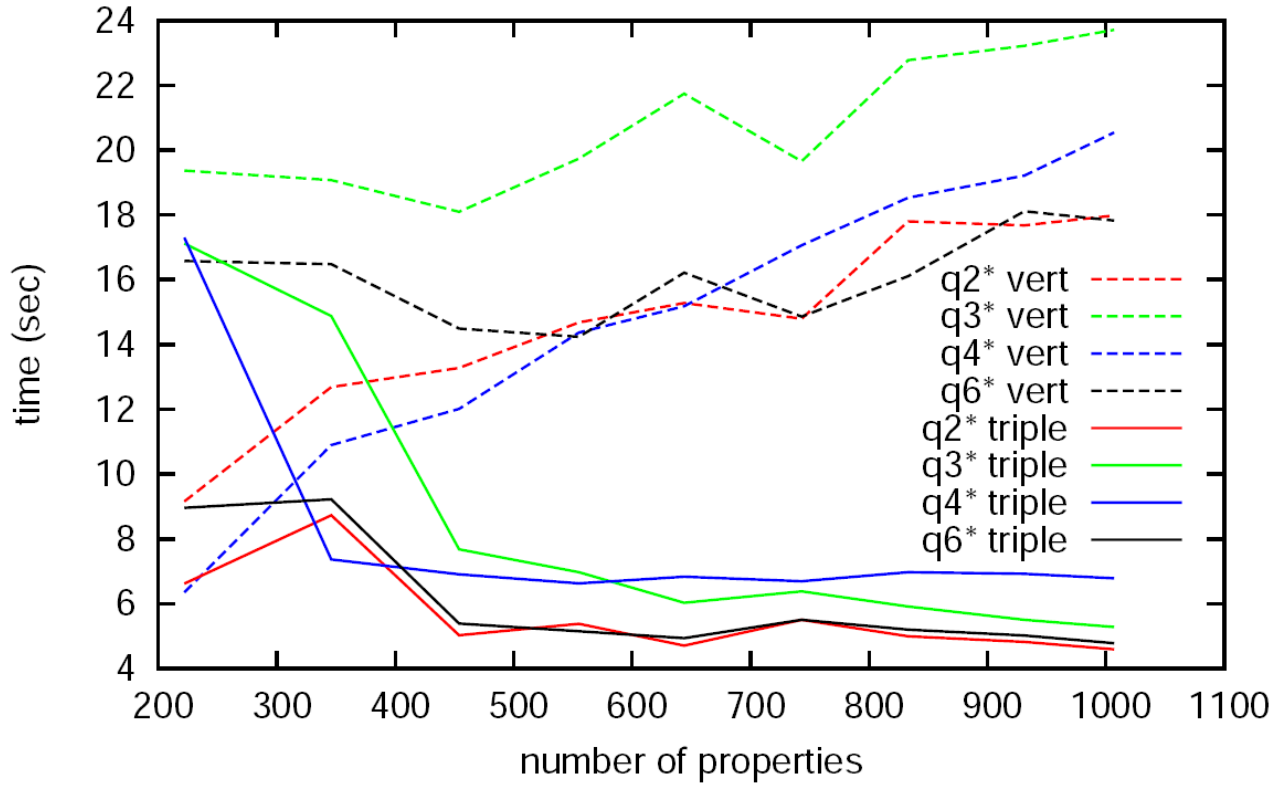
First Experiment - Triple Store vs Vertically Partitioned

	store	cluster	time	q1	q2	q2*	q3	q3*	q4	q4*	q5	q6	q6*	q7	q8	G	G*	$\frac{G^*}{G}$
DBX	triple	SPO	real	4.29	42.61	93.11	34.86	97.92	8.02	6.12	11.70	89.11	142.10	1.34	14.47	13.2	21.1	1.6
			user	4.29	33.31	68.88	34.16	95.11	8.02	6.10	11.68	74.96	120.36	1.27	10.58	12.3	19.0	1.5
		PSO	real	1.72	40.18	38.35	45.65	67.32	3.22	2.49	10.61	57.52	63.04	1.42	12.14	9.8	13.6	1.4
			user	1.72	40.17	38.35	45.64	66.85	3.22	2.47	10.60	57.33	63.03	1.34	8.02	9.7	13.1	1.4
	vert.	SO	real	1.55	39.62	74.85	45.17	94.59	6.12	14.18	5.69	45.57	154.81	1.25	11.55	9.1	17.7	**1.9**
			user	1.55	39.61	74.83	45.16	94.09	6.12	14.15	5.67	45.56	153.08	1.18	7.49	9.1	17.0	**1.9**
MonetDB	triple	SPO	real	1.53	3.50	3.63	5.28	17.54	1.68	1.98	2.77	8.37	7.33	1.82	4.76	2.9	3.7	1.3
			user	1.36	2.73	2.91	4.33	15.40	1.41	1.65	2.30	6.20	5.70	1.65	3.75	2.4	3.1	1.3
		PSO	real	0.78	2.80	2.83	4.36	12.59	1.70	1.97	1.44	5.67	4.59	0.18	5.23	1.5	2.4	1.6
			user	0.69	2.31	2.31	3.69	10.54	1.59	1.86	1.16	3.80	3.65	0.17	3.60	1.3	2.0	1.5
	vert.	SO	real	0.79	1.50	5.50	2.64	14.01	0.50	2.57	1.29	4.65	11.51	0.06	5.05	0.9	2.0	**2.2**
			user	0.68	1.44	5.20	2.52	13.25	0.48	2.40	1.03	4.40	11.23	0.06	4.20	0.8	1.9	**2.4**
C-Store	vert.	SO	real	0.59	1.35	–	4.08	–	1.52	–	12.95	12.04	–	0.77	–	2.4	–	–
			user	0.49	1.17	–	3.45	–	1.28	–	11.67	10.49	–	0.34	–	1.9	–	–

First Experiment - Triple Store vs Vertically Partitioned cont'd

The vertical approach outperforms the triple storage on allmost all queries without a star.
What happens if the number of properties is further increased?

First Experiment - Triple Store vs Vertically Partitioned cont'd

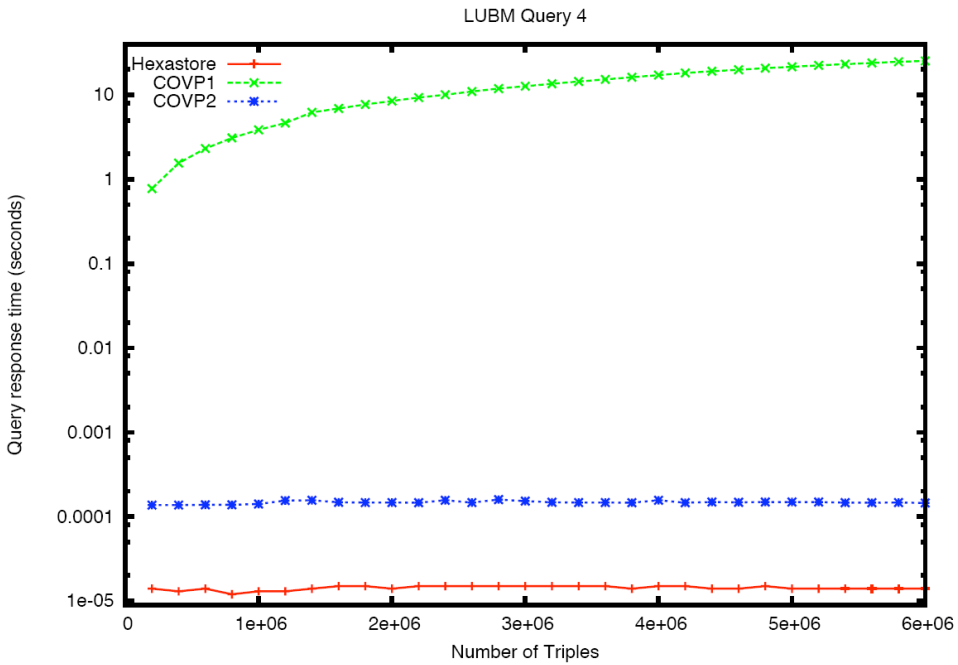


Second Experiment - Hexastore vs Vertically Partitioned

Second Experiment - Hexastore vs Vertically Partitioned

Results:

- Hexastore outperforms vert part approach.



Second Experiment - Hexastore vs Vertically Partitioned

But:

- All triples were loaded in the main memory beforehand

Not all swans are white!

Every storage scheme has its weaknesses. The triple store with elaborated usage of the “right“ indeces seems to have the best performance regarding scalability.

Thanks for your attention!!

References