Scalable Semantic Web Data Management Using Vertical Partitioning

Daniel J. Abadi, Adam Marcus, Samuel R. Madden, Kate Hollenbach
(MIT)

VLDB 2007

Presented By: Pragnya Addala

Introduction

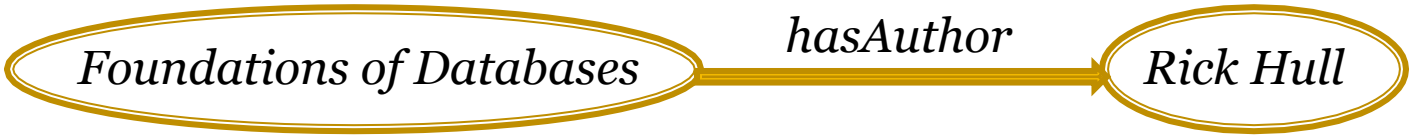
- Motivation:
  - Semantic Web necessitates high-performance data management tools
  - Current state-of-the-art RDF databases (triple-stores) perform poorly
  - Solution suggested in previous works
    - Property table – has undesirable features
  - Solution proposed
    - Vertically partitioning RDF data
    - Extension to column-oriented DBMS

Introduction

- Goal:
  - Achieve scalability and performance in triple stores
- Methodology
  - Current state-of-the-art: Investigate approaches in RDBMS and suggested techniques like "property tables"
  - Explore benefits of vertical partitioning and column-store
  - Compare performance of different RDF storage schemes on a real world RDF dataset

RDF Triples

- Semantic breakdown
  - "Rick Hull wrote Foundations of Databases."
- Representation
  - Graph



  - Statement

    <"Foundations of Databases", hasAuthor, "Rick Hull">

  - XML format

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
          <rdf:Description rdf:about="Foundations of Databases>
                    <hasAuthor>Rick Hull</hasAuthor>
          </rdf:Description>
</rdf:RDF>
```

RDF in RDBMSs - Issues

- Relation database
  - 3-column schema
- Issues:
  - Performance (Many self-joins)
  - 1 Massive triples table

```
SELECT ?title
FROM table
WHERE { ?book author ''Fox, Joe''
        ?book copyright ''2001''
        ?book title ?title }
```

```
SELECT C.obj.
FROM TRIPLES AS A,
     TRIPLES AS B,
     TRIPLES AS C
WHERE A.subj. = B.subj.
  AND B.subj. = C.subj.
  AND A.prop. = 'copyright'
  AND A.obj. = ''2001''
  AND B.prop. = 'author'
  AND B.obj. = ''Fox, Joe''
  AND C.prop. = 'title'
```

Subj.	Prop.	Obj.
ID1	type	BookType
ID1	title	"XYZ"
ID1	author	"Fox, Joe"
ID1	copyright	"2001"
ID2	type	CDType
ID2	title	"ABC"
ID2	artist	"Orr, Tim"
ID2	copyright	"1985"
ID2	language	"French"
ID3	type	BookType
ID3	title	"MNO"
ID3	language	"English"
ID4	type	DVDType
ID4	title	"DEF"
ID5	type	CDType
ID5	title	"GHI"
ID5	copyright	"1995"
ID6	type	BookType
ID6	copyright	"2004"

Property Tables

- Goal: Speed up queries over triple-stores
- Idea:
  - Cluster triples containing properties defined over similar subjects
    - Example: “title”, “author”, “copyright”
  - Exploit the type property of subjects to cluster similar sets of subjects
    - Example: Books, journals, CDs, etc.
- Reduces number of self-joins

Clustered Property Tables

Subj.	Prop.	Obj.
ID1	type	
ID1	title	"XYZ"
ID1	author	"Fox, Joe"
ID1	copyright	"2001"
ID2	type	CDType
ID2	title	"ABC"
ID2	artist	"Orr, Tim"
ID2	copyright	"1985"
ID2	language	"French"
ID3	type	BookType
ID3	title	"MNO"
ID3	language	"English"
ID4	type	DVDType
ID4	title	"DEF"
ID5	type	CDType
ID5	title	"GHI"
ID5	copyright	"1995"
ID6	type	BookType
ID6	copyright	"2004"

Exploit clusters of properties that tend to be defined together (properties for similar subjects)

Property Table

Subj.	Type	Title	copyright
ID1	BookType	"XYZ"	"2001"
ID2	CDType	"ABC"	"1985"
ID3	BookType	"MNP"	NULL
ID4	DVDType	"DEF"	NULL
ID5	CDType	"GHI"	"1995"
ID6	BookType	NULL	"2004"

Left-Over Triples

Subj.	Prop.	Obj.
ID1	author	"Fox, Joe"
ID2	artist	"Orr, Tim"
ID2	language	"French"
ID3	language	"English"

Property-Class Tables

Subj.	Prop.	Obj.
ID1	type	BookType
ID1	title	"XYZ"
ID1	author	"Fox, Joe"
ID1	copyright	"2001"
ID2	type	CDType
ID2	title	"ABC"
ID2	artist	"Orr, Tim"
ID2	copyright	"1985"
ID2	language	"French"
ID3	type	BookType
ID3	title	"MNO"
ID3	language	"English"
ID4	type	DVDType
ID4	title	"DEF"
ID5	type	CDType
ID5	title	"GHI"
ID5	copyright	"1995"
ID6	type	BookType
ID6	copyright	"2004"

Exploit type of property to cluster similar sets of subjects

Class: BookType

Subj.	Title	Author	copyright
ID1	"XYZ"	"Fox, Joe"	"2001"
ID3	"MNP"	NULL	NULL
ID6	NULL	NULL	"2004"

Class: CDType

Subj.	Title	Artist	copyright
ID2	"ABC"	"Orr, Tim"	"1985"
ID5	"GHI"	NULL	"1995"

Left-Over Triples

Subj.	Prop.	Obj.
ID2	language	"French"
ID3	language	"English"
ID4	type	DVDType
ID4	title	"DEF"

Property Tables - Issues

- Complex to design
- If table is made narrow with fewer property columns
  - table is less sparse
  - query confined to one property table is reduced -> more unions and joins
- If table is made wider including more property columns
  - number of unions and joins decreases
  - more NULLs (more sparse) -> more wastage of space
- Further complexity is added by multi-valued attributes
  - cannot be added in the same table with other attributes
- Queries that do not select on property class type
  - problematic for property-class tables
- Queries that have unspecified property values
  - problematic for clustered property tables

Vertically Partitioned Approach

- Idea: One table per property
  - Column 1 –subjects that define the property
  - Column 2 – object values for those subjects
- Each table sorted by subject
  - Particular subjects can be located quickly
  - Fast merge joins

Vertically Partitioned Approach

Subj.	Prop.	Obj.
ID1	type	BookType
ID1	title	"XYZ"
ID1	author	"Fox, Joe"
ID1	copyright	"2001"
ID2	type	CDType
ID2	title	"ABC"
ID2	artist	"Orr, Tim"
ID2	copyright	"1985"
ID2	language	"French"
ID3	type	BookType
ID3	title	"MNO"
ID3	language	"English"
ID4	type	DVDType
ID4	title	"DEF"
ID5	type	CDType
ID5	title	"GHI"
ID5	copyright	"1995"
ID6	type	BookType
ID6	copyright	"2004"

1 table per property
Column 1 – subject
Column 2 - object

Type	
ID1	BookType
ID2	CDType
ID3	BookType
ID4	DVDType
ID5	CDType
ID6	BookType

Author	
ID1	"Fox, Joe"

Title	
ID1	"XYZ"
ID2	"ABC"
ID3	"MNO"
ID4	"DEF"
ID5	"GHI"

Artist	
ID2	"Orr, Tim"

Copyright	
ID1	"2001"
ID2	"1985"
ID5	"1995"
ID6	"2004"

Language	
ID2	"French"
ID3	"English"

Vertically Partitioned Approach

- Advantages
  - Multi-values attributes supported
  - Support for heterogeneous records
    - Null data not stored
  - No clustering algorithm
  - Only accessed properties are read
  - Fewer unions
    - Data for particular property is located in same table
  - Fast Joins
    - Simple, fast (linear) merge joins

Author

ID1	"Fox, Joe"
ID1	"Green, John"

Extending a Column-Oriented DBMS

- Column-store is a natural storage layer to use vertical partitioning
- Advantages
  - Tuple headers stored separately
    - 35 bytes in Postgres vs. 8 bytes in C-Store
  - Column-oriented data compression
    - Run-length encoding (ex. 1,1,1,2,2 -> 1X3, 2X2)
  - Do not necessarily have to store the subject column
  - Carefully optimized merge-join code
    - Prefetching

Materialized Path Expression

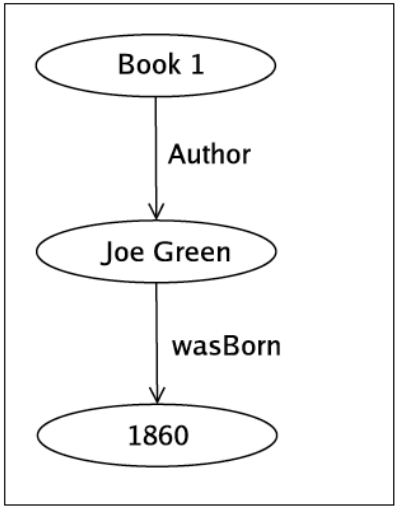
- Subject-object joins are replaced by cheaper subject-subject joins
- We can add a new column representing materialized path expression
- Inference queries are a common operation on Semantic Web data which can be accelerated using this method.

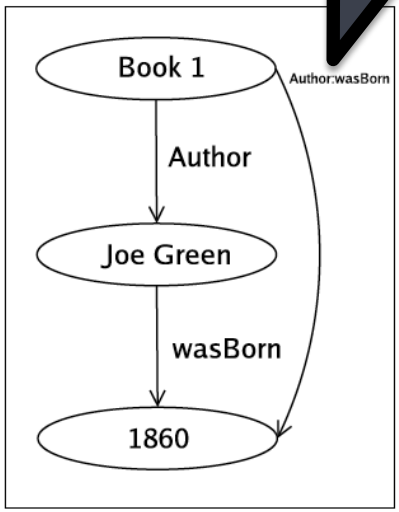
Materialized Path Expression

```
SELECT B.subj
FROM triples AS A, triples AS B
WHERE A.prop = wasBorn
AND A.obj = ‘‘1860’’
AND A.subj = B.obj
AND B.prop = ‘‘Author’’
```

```
SELECT A.subj
FROM proptable AS A,
WHERE A.author:wasBorn = ‘‘1860’’
```

Materialized path expression

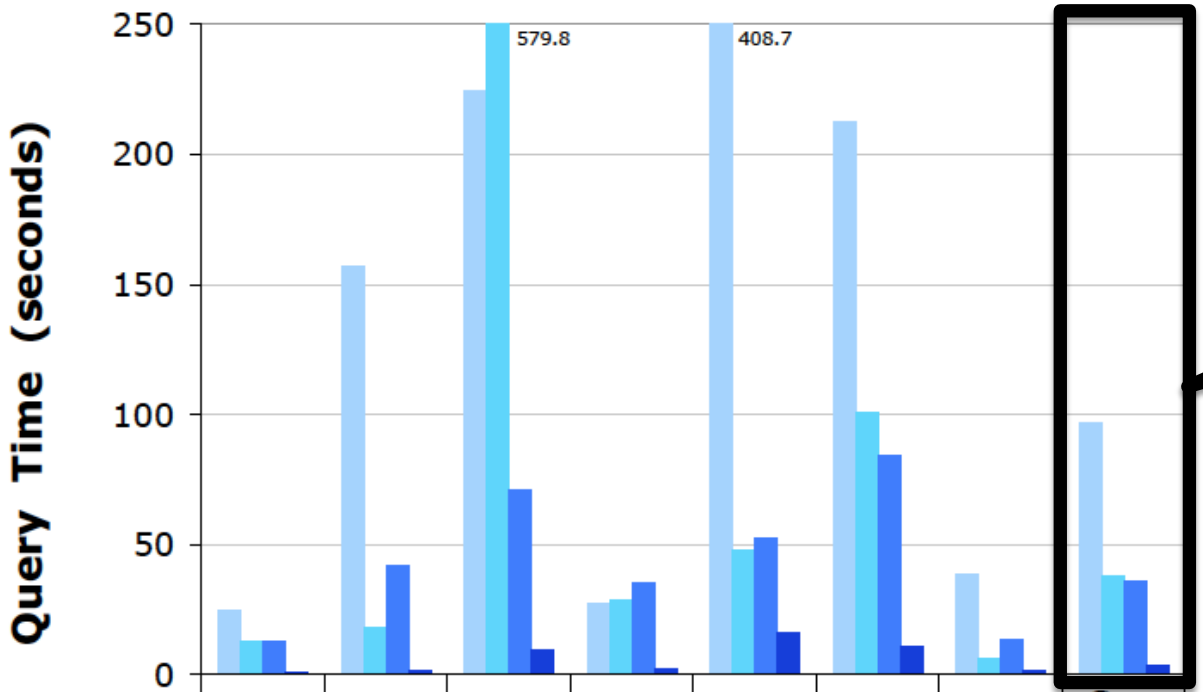




Experimental Evaluation

- Barton Libraries Dataset
- Longwell Queries
  - Calculating counts
  - Filtering
  - Inference

Results – Performance Numbers



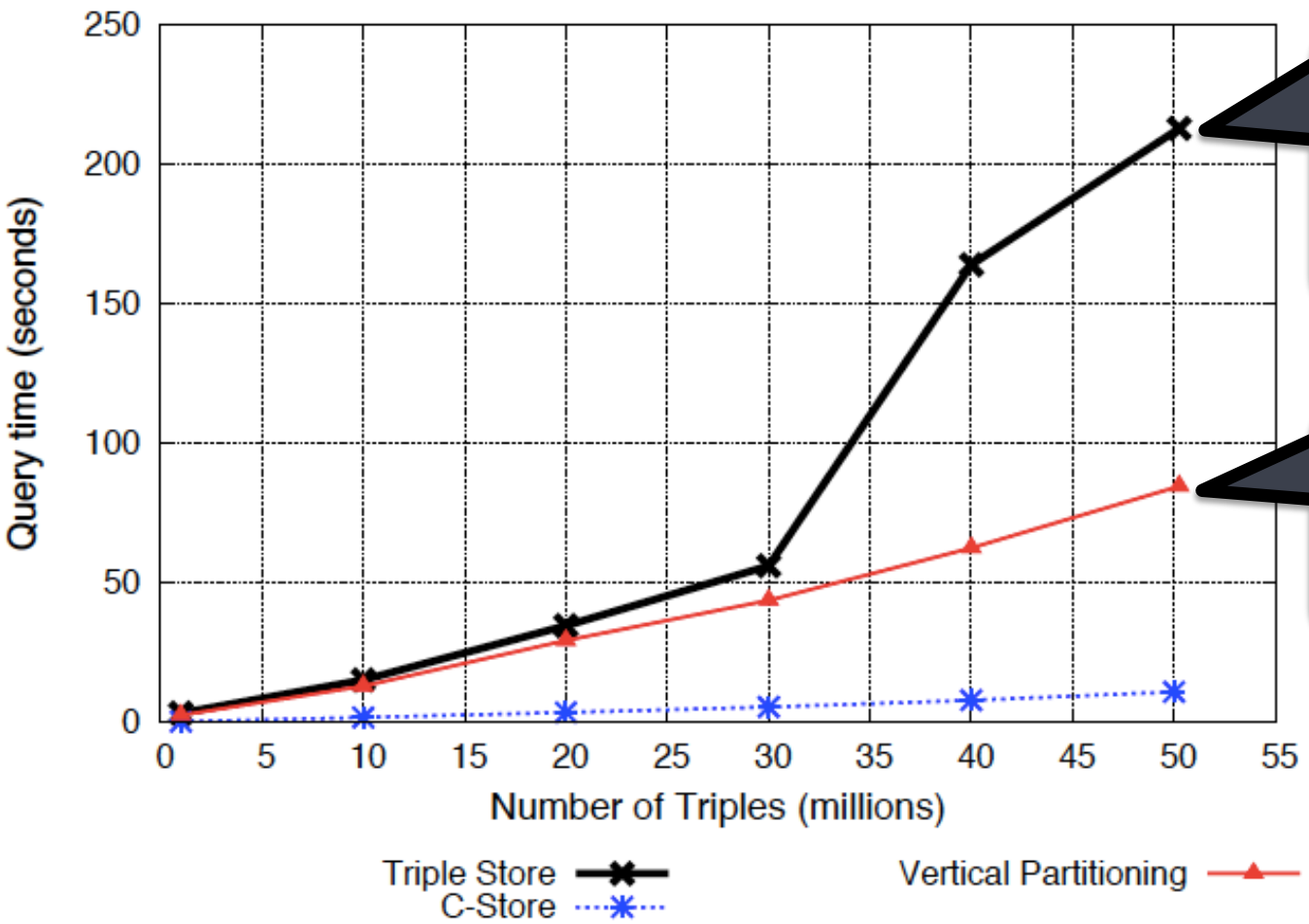
	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Geo. Mean
Triple Store	24.63	157	224.3	27.67	408.7	212.7	38.37	97
Prop. Table	12.66	18.37	579.8	28.54	47.85	101	6.1	38
Vert. Part.	12.66	41.7	71.3	35.49	52.34	84.6	13.25	36
C-Store	0.66	1.64	9.28	2.24	15.88	10.81	1.44	3

Property Table and Vertical Partitioning = 2x-3x Triple-Store

C-Store added another factor of 10 improvement

Results - Scalability

- Query 6 performance as number of triples scale



Super-Linear (requires sorting of intermediate results after 3 selections and before join)

Linear

Results – Materialized Path Expressions

- Replace
  - Subject-object joins -> subject-subject joins

	Q5	Q6
Property Table	39.49 (17.5% faster)	62.6 (38% faster)
Vertical Partitioning	4.42 (92% faster)	65.84 (22% faster)
C-Store	2.57 (84% faster)	2.70 (75% faster)

Results – Effect of Further Widening

Query	Wide Property Table	Property Table % slowdown
Q1	60.91	381%
Q2	33.93	85%
Q3	584.84	1%
Q4	44.96	58%
Q5	76.34	60%
Q6	154.33	53%
Q7	24.25	298%

Conclusion

- Semantic Web users require fast responses to queries
- RDF triples store scales extremely poorly because multiple self joins are required
- Poorly-selected property table is BAD
- Propose vertically partitioning tables
  - achieves similar performance in a row-oriented database
  - SIMPLER to implement
  - very good with column-oriented database

Comments/Discussion

- Promising results for reads, but writes not considered. Ideas?
- What about load times?