Hexastore:
Sextuple Indexing for Semantic Web Data Management

Cathrin Weiss [a], Panagiotis Karras [b], and Abraham Bernstein [a]
[b] National University of Singapore
[a] University of Zurich

Presented By

Mohamed Sabri

Motivation:

- The vision of the Semantic Web is to allow everybody to publish *linked,* machine-readable, data.
- RDF is the standard data model for this vision.
- Most existing RDF stores suffer from either scalability issues or a specialization of their architecture for special-type queries, or both.
- There is a need for an RDF storage scheme that *uses the triple nature of RDF as an asset* to overcome the above limitations.

Overview:

- The proposed framework is based on the idea of indexing the RDF data in six possible ways, one *for each possible ordering* of the three RDF elements; 3! =6.

- This format allows for quick and scalable *general-purpose* query processing.

- It offers up to *five orders of magnitude* performance improvement compared to previous approaches.

- But, at the price of a *worst-case five-fold increase* in index space.

Previous Approaches:

- Relational-based approaches:
  - Utilize traditional relational database as a backend (Jena, IBM, Oracle, 3Store,…etc).
  - Statement-based queries can be satisfactorily processed by such systems. (A statement-based query lacks one or two parts of a triple)
  - RDF triples were traditionally stored in a giant *triples table*, causing serious scalability problems.
  - Recent approaches tries to construct tables that gathered many properties together as attributes. (example next…)

Example (IBM):

entry	*spill*	$pred_1$	val_1	$pred_2$	val_2	$pred_3$	val_3	...	$pred_k$	val_k
Charles Flint	0	died	1934	born	1850	founder	IBM	...	*null*	*null*
Larry Page	0	board	Google	born	1973	founder	Google	...	home	Palo Alto
Android	1	developer	Google	version	4.1	kernel	Linux	...	preceded	4.0
Android	1	*null*	*null*	*null*	*null*	graphics	OpenGL	...	*null*	*null*
Google	0	industry	lid:1	employees	54,604	*null*	*null*	...	HQ	Mtn View
IBM	0	industry	lid:2	employees	433,362	*null*	*null*	...	HQ	Armonk

l_id	*elm*
lid:1	Software
lid:1	Internet
lid:2	Software
lid:2	Hardware
lid:2	Services

← Separate table for multi-valued properties.

Example (IBM):

entry	*spill*	$pred_1$	val_1	$pred_2$	val_2	$pred_3$	val_3	...	$pred_k$	val_k
Charles Flint	0	died	1934	born	1850	founder	IBM	...	*null*	*null*
Larry Page	0	board	Google	born	1973	founder	Google	...	home	Palo Alto
Android	1	developer	Google	version	4.1	kernel	Linux	...	preceded	4.0
Android	1	*null*	*null*	*null*	*null*	graphics	OpenGL	...	*null*	*null*
Google	0	industry	lid:1	employees	54,604	*null*	*null*	...	HQ	Mtn View
IBM	0	industry	lid:2	employees	433,362	*null*	*null*	...	HQ	Armonk

l_id	*elm*
lid:1	Software
lid:1	Internet
lid:2	Software
lid:2	Hardware
lid:2	Services

← Separate table for multi-valued properties.

- *"However, imposing structure where it does not naturally exist results into sparse representation with many NULL values in the formed property tables."*

Previous Approaches (cont'd):

- Multiple-indexing approaches (Harth and Decker; Wood et al.):
  - Constructs six indices that cover all $2^4 = 16$ possible access patterns of quads in the form {s,p,o,c}, where c a is the context of triple {s, p, o}.
  - Thus, it is also oriented towards simple statement-based queries.

No	Access pattern	No	Access pattern
1	(?:?:?:?)	9	(s:?:o:c)
2	(s:?:?:?)	10	(?:?:o:c)
3	(s:p:?:?)	11	(?:?:o:?)
4	(s:p:o:?)	12	(?:?:?:c)
5	(s:p:o:c)	13	(s:?:?:c)
6	(?:p:?:?)	14	(s:p:?:c)
7	(?:p:o:?)	15	(?:p:?:c)
8	(?:p:o:c)	16	(s:?:o:?)

Previous Approaches (cont'd):

- The Vertical-partitioning approach (Abadi et al):
  - *Triples table* is rewritten into n two-column tables, one table per property.
  - Multi-valued subjects are still represented by multiple rows!
  - This model is strictly property-oriented!
  - Problem with *non-property-bound queries* (All two-column tables will have to be queried)

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"

Previous Approaches (cont'd):

- The Vertical-partitioning approach (Abadi et al):
  - *Triples table* is rewritten into n two-column tables, one table per property.
  - Multi-valued subjects are still represented by multiple rows!
  - This model is strictly property-oriented!
  - Problem with *non-property-bound queries* (All two-column tables will have to be queried)

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"

Examples of non-property-bound queries:

1) Ask what kind of relationship, if any, a certain person has to MIT:

```
SELECT A.property
FROM triples AS A
WHERE A.subj = ID2 AND A.obj = 'MIT'
```

2) Ask for people who have the same relationship to Stanford as a certain person has to Yale:

```
SELECT B.subj
FROM triples AS A, triples AS B,
WHERE A.subj = ID1 AND A.obj = 'Yale'
AND A.property = B.property AND B.obj =
'Stanford'
```

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'

Examples of non-property-bound queries:

1) Ask what kind of relationship, if any, a certain person has to MIT:

```
SELECT A.property
FROM triples AS A
WHERE A.subj = ID2 AND A.obj = 'MIT'
```

2) Ask for people who have the same relationship to Stanford as a certain person has to Yale:

```
SELECT B.subj
FROM triples AS A, triples AS B,
WHERE A.subj = ID1 AND A.obj = 'Yale'
AND A.property = B.property AND B.obj =
'Stanford'
```

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'

Hexastore:

- Hexastore extends the last two approaches (Multiple-indexing and Vertical-partitioning) and takes the idea further “to its logical conclusion”!

- The resulting solution does not discriminate against any RDF element; each RDF element type deserves to have special index structures built around it.

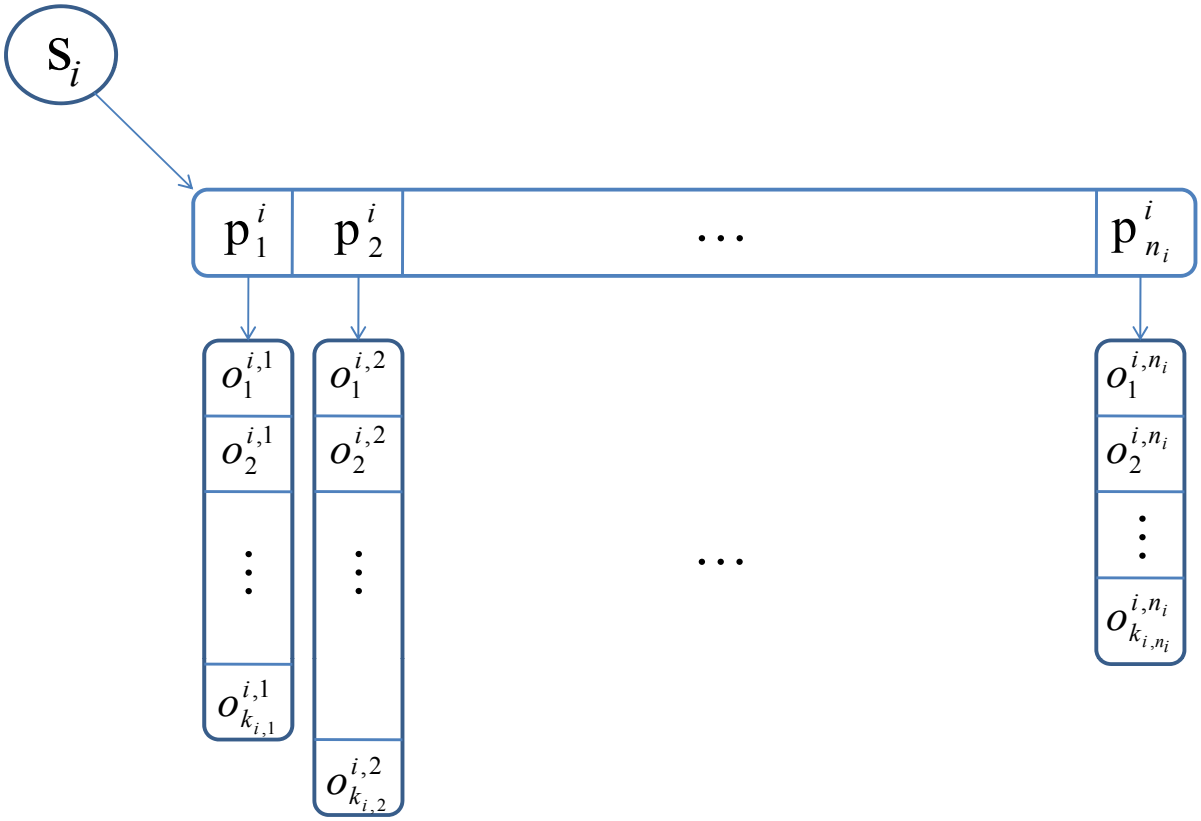
- Every possible ordering of the three elements in an indexing scheme is materialized;

  sp**o**, so**p**, ps**o**, po**s**, os**p**, and op**s**.

- Employs a dictionary encoding; maps string URIs to integer identifiers.

Hexastore (cont'd):

- Example; **spo** index, the indexing that groups data into subject-headed divisions with property vectors and lists of objects per vector…



Advantages of Hexastore:

- Concise and efficient handling of multi-valued resources.
- Avoidance of NULLs.
- No ad-hoc choices needed
  - The architecture of a Hexastore is uniquely and deterministically defined by the RDF data at hand.
- Reduced I/O cost
  - …accesses only those items that are relevant to the query… property-based approaches need to access all property tables in a store for queries that are not bound by property.
- All first-step pairwise joins are fast merge-joins.
  - The keys of resources in all vectors and lists used in a Hexastore are sorted.
- Reduction of unions and joins.

IBM Example (revisited):

entry	*spill*	$pred_1$	val_1	$pred_2$	val_2	$pred_3$	val_3	...	$pred_k$	val_k
Charles Flint	0	died	1934	born	1850	founder	IBM	...	*null*	*null*
Larry Page	0	board	Google	born	1973	founder	Google	...	home	Palo Alto
Android	1	developer	Google	version	4.1	kernel	Linux	...	preceded	4.0
Android	1	*null*	*null*	*null*	*null*	graphics	OpenGL	...	*null*	*null*
Google	0	industry	lid:1	employees	54,604	*null*	*null*	...	HQ	Mtn View
IBM	0	industry	lid:2	employees	433,362	*null*	*null*	...	HQ	Armonk

l_id	*elm*
lid:1	Software
lid:1	Internet
lid:2	Software
lid:2	Hardware
lid:2	Services

← Separate table for multi-valued properties.

Advantages of Hexastore:

- Concise and efficient handling of multi-valued resources.
- Avoidance of NULLs.
- No ad-hoc choices needed
  - The architecture of a Hexastore is uniquely and deterministically defined by the RDF data at hand.
- Reduced I/O cost
  - …accesses only those items that are relevant to the query… property-based approaches need to access all property tables in a store for queries that are not bound by property.
- All first-step pairwise joins are fast merge-joins.
  - The keys of resources in all vectors and lists used in a Hexastore are sorted.
- Reduction of unions and joins.

Drawbacks of Hexastore:

- The main drawback of the Hexastore is its worst-case five-fold increase in storage space
  - However, the increase in storage is guaranteed and deterministic; thus, it shows predictable storage requirements that can be planned ahead.

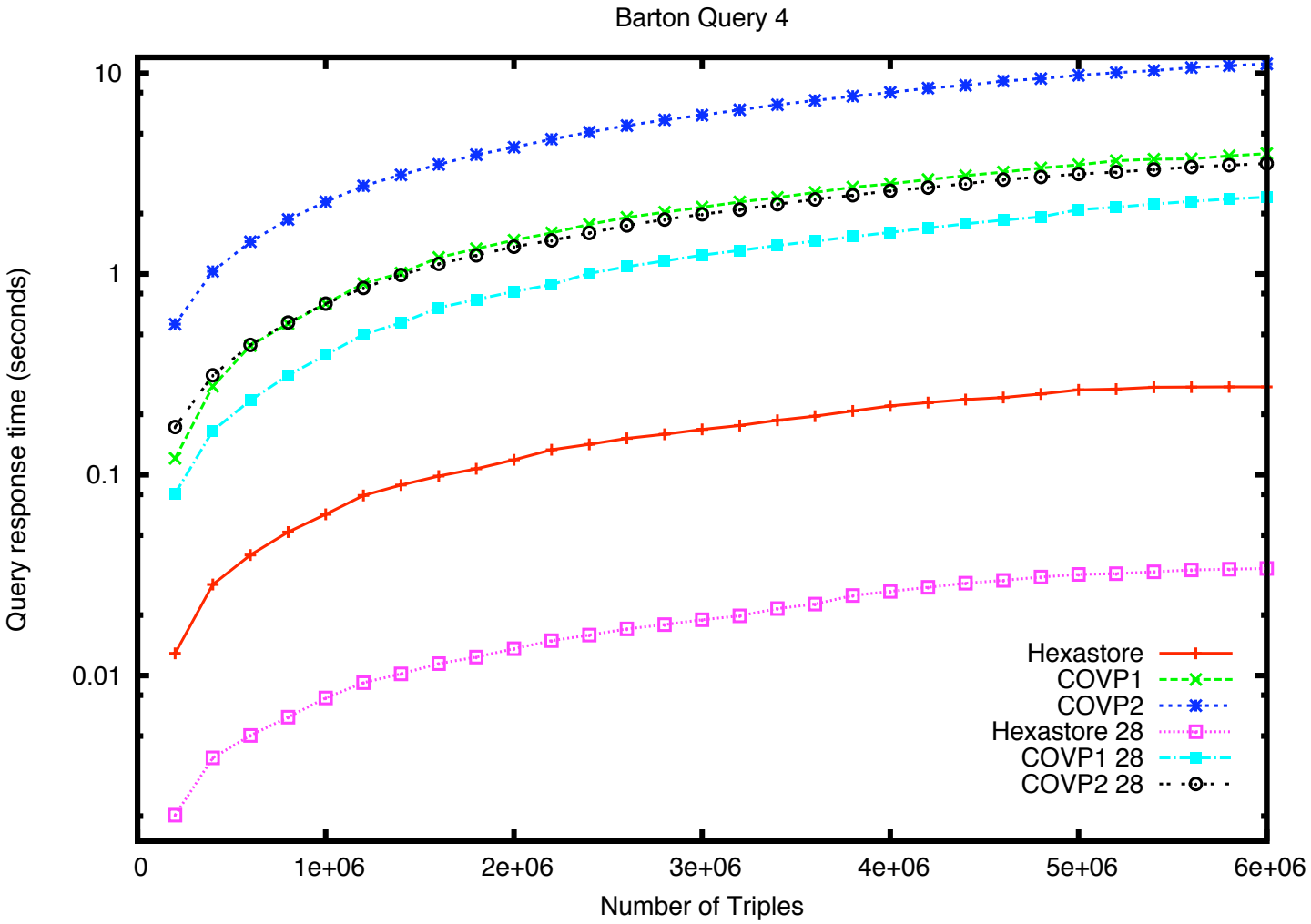
- Another deficiency of the Hexastore appears when it comes to handling updates and insertions
  - However, existing RDF applications do not involve massive updates.

Experimental Evaluation:

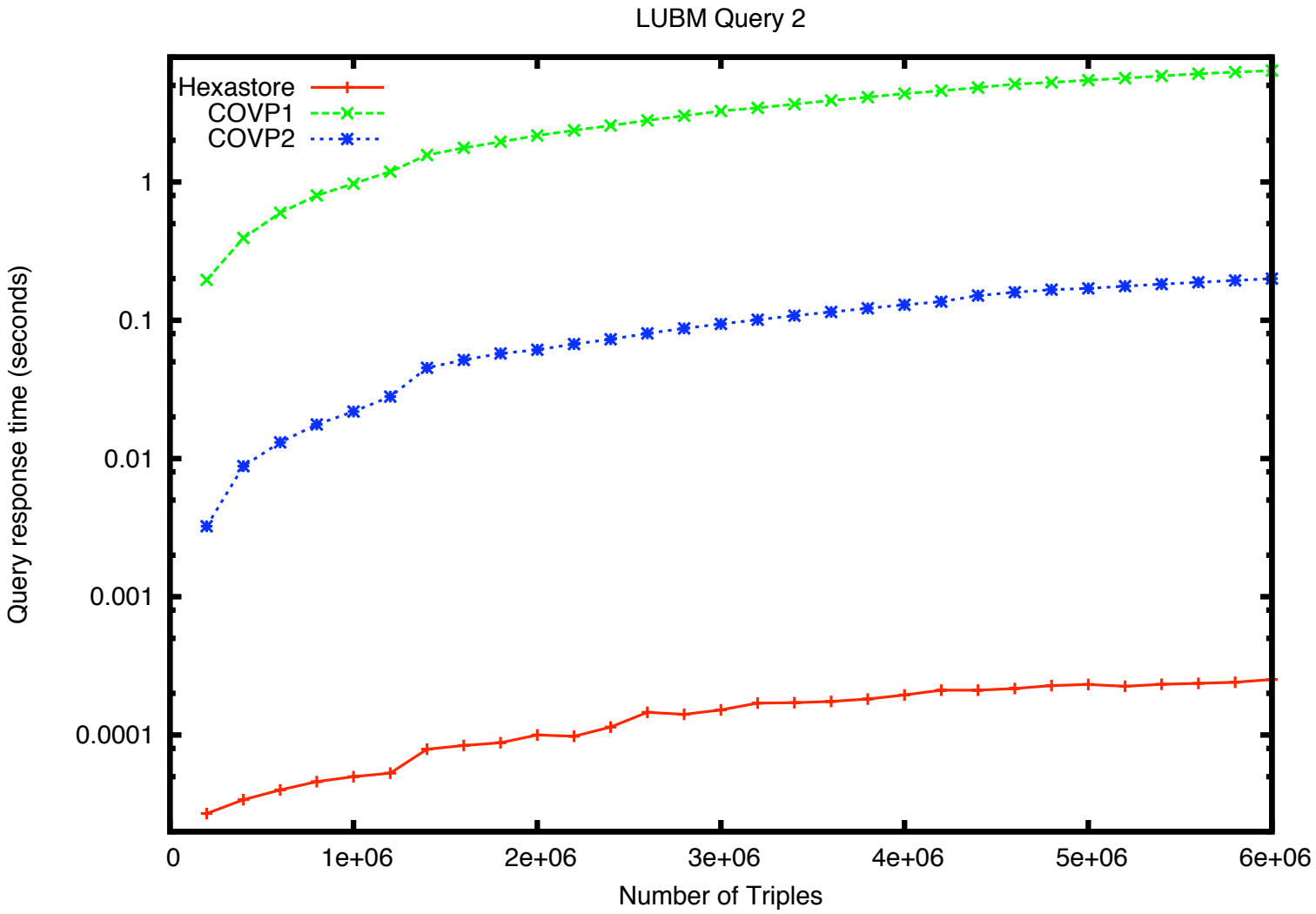
- Compare the performance of Hexastore to…
  - the authors' implementation of the column-oriented vertical-partitioning (COVP1) method, i.e. pso index;
  - and the two-index version COVP2, i.e. pso and pos indexes.

- Using two data sets:
  1. Barton: a real data set, consisting of 61,233,949 triples, having a total of 285 unique properties.
  2. LUBM: a synthetic data set, consisting of 6,865,225 triples, having a total of 18 unique properties.

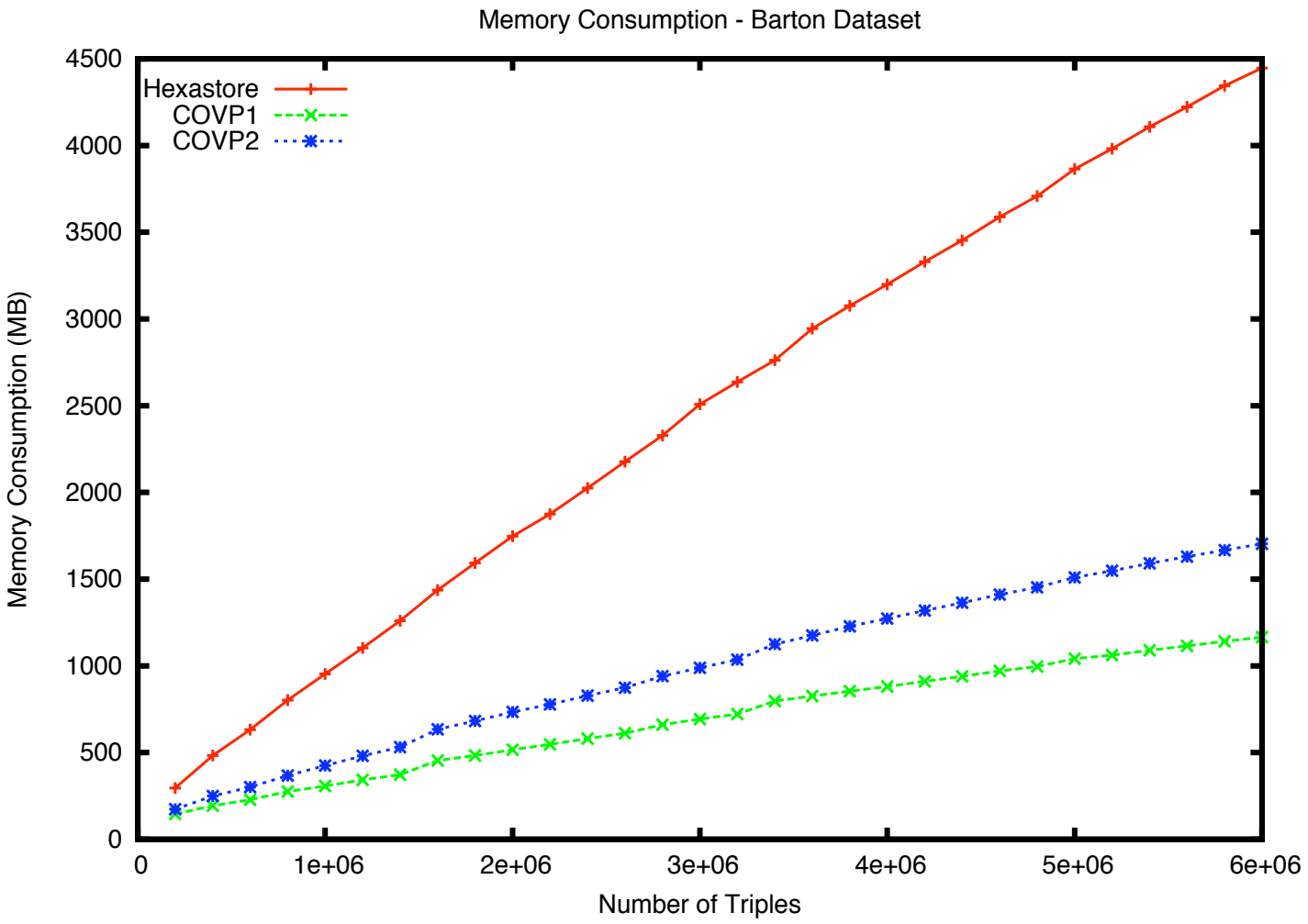
Experimental Evaluation (Barton, query performance):



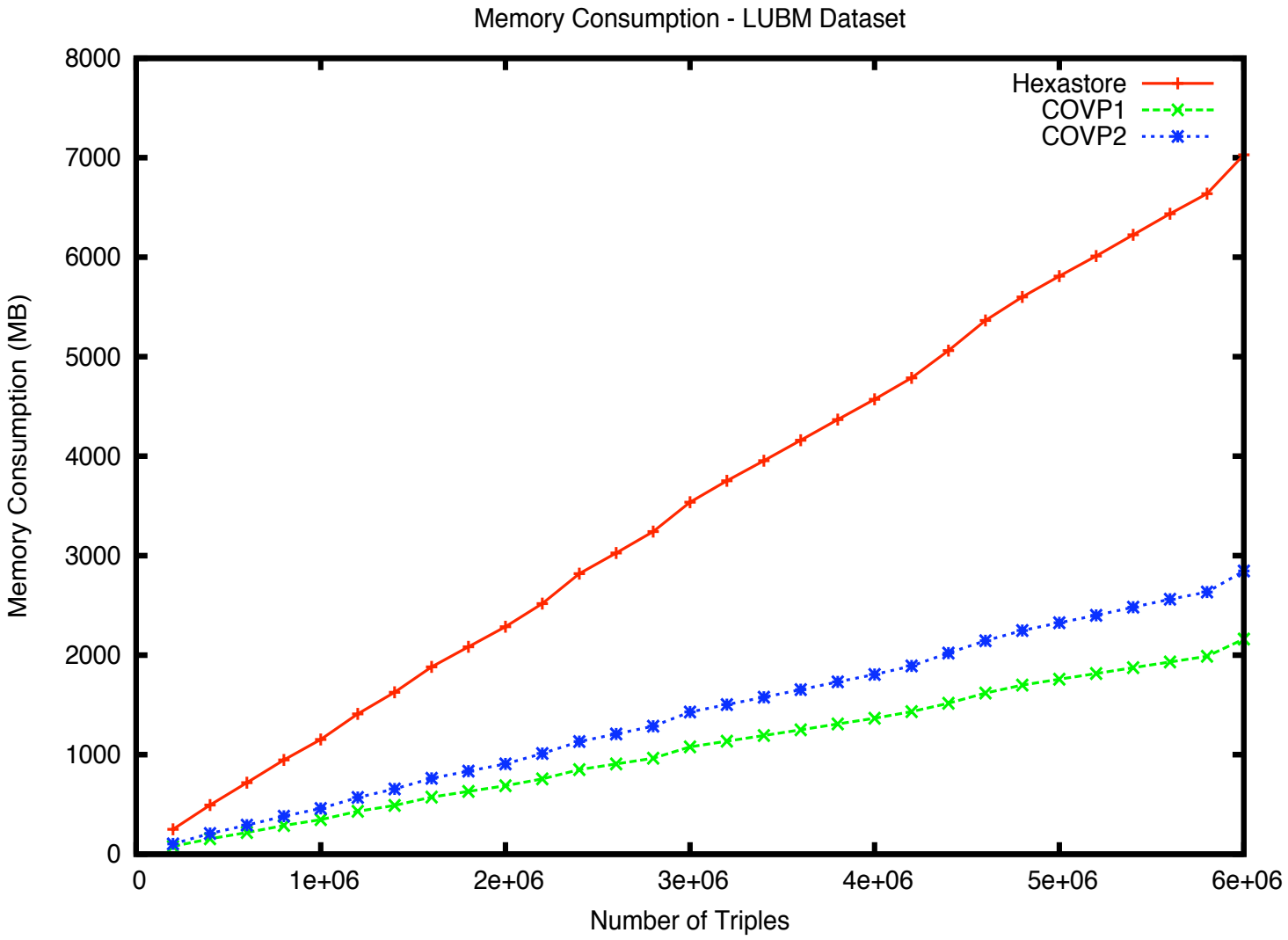
Experimental Evaluation (LUBM, query performance):



Experimental Evaluation (Barton, memory usage):



Experimental Evaluation (LUBM, memory usage):



Conclusion & Future Work:

- Hexastore allows for efficient and scalable general-purpose query processing.
- It offers advantages of up to five orders of magnitude in comparison to previous approaches.
- The Hexastore is concise in its individual representations, but the replication of these representations in a five-fold manner may exceed the available storage capacity.
- Some indices may be rarely used in a specific workload; investigate the automatic/adaptive selection of the most suitable indices to materialize.
- Apply Database Cracking techniques to Hexastore (incremental and adaptive indexing abilities).

Thank You!