Rya: A Scalable RDF Triple Store for the Clouds

R. Punnose[1], **Adina Crăiniceanu[2]**, D. Rapp[3]

1-Proteus Technologies, 2-U.S. Naval Academy, 3-LTS

Presentation at Cloud Intelligence Workshop 2012, a VLDB 2012 workshop

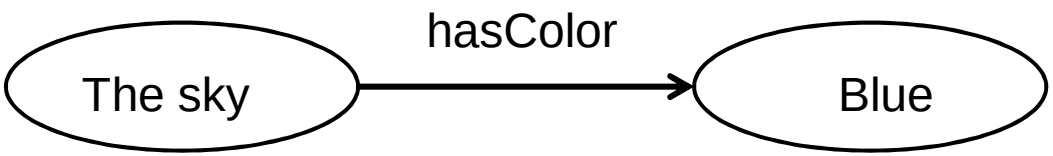
RDF Data

- Increasingly popular
- Based on making statements about resources
  - Statements are formed as triples

    (subject-predicate-object)
- Example, “The sky has the color blue”
  - Subject = The sky
  - Predicate = has color
  - Object = blue

Why RDF?

- W3C standard
- Large community/tool support
- Easy to understand
- Intrinsically represents a labeled, directed graph



- Unstructured
  - Though with RDFS/OWL, can add structure

Problem * * * *

Why Not RDF?

- Storage
  - Stores can be large for small amounts of data
- Speed
  - Slow to answer simple questions
- Scale
  - Not easy to scale with size of data

Rya –Distributed RDF Triple Store

- Smartly store RDF data in Accumulo
  - Scalability
  - Load balance
- Build on the OpenRDF interface implementation for SPARQL
  - Fast queries

Problem * * * *

Outline

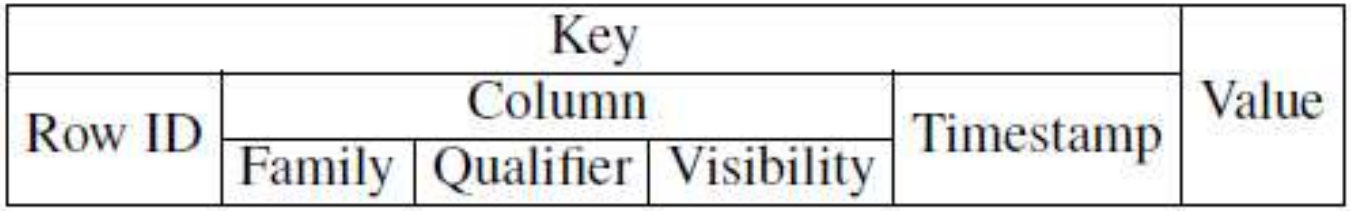
- Problem
- Background
- Rya
  - Triple index
  - Performance enhancements
- Experimental results
- Conclusions and future work

OpenRDF Sesame

- Utilities to parse, store, and query RDF data
- Supports SPARQL
- Ex: SELECT ?x WHERE {
  ?x rdf:type Faculty .
  ?x degreeFrom Cornell . }
- SPARQL queries evaluated based on triple patterns
  - Ex: (*, rdf:type, Faculty)

Accumulo

- Google BigTable implementation

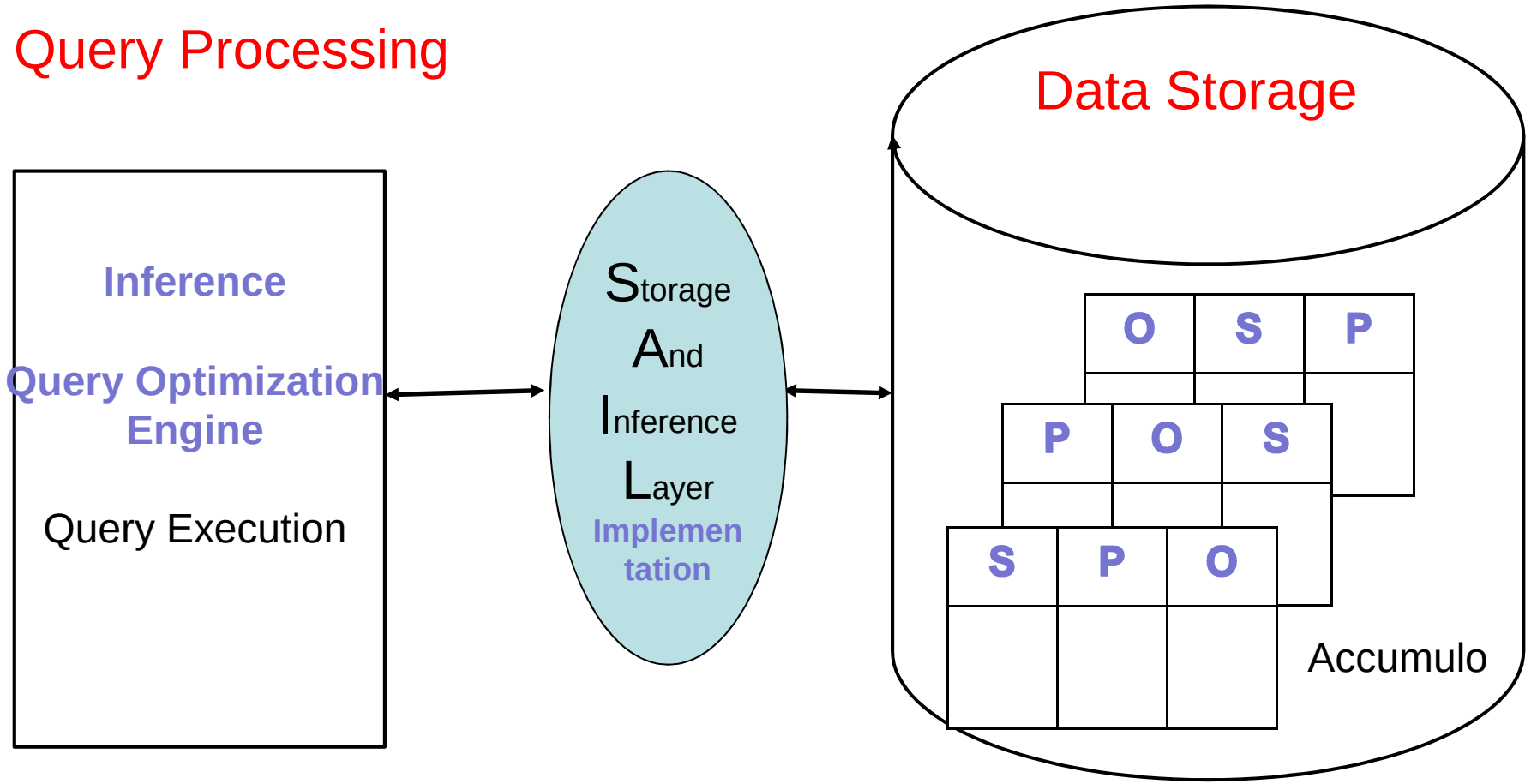


  - Compressed, Distributed, Scalable
- Adds security, row level authentication/ visibility, etc
- The Accumulo store acts as persistence and query backend to OpenRDF

Outline

- Problem
- Background
- Rya
  - Triple index
  - Performance enhancements
- Experimental results
- Conclusions and future work

Architectural Overview - Rya



Triple Table Index

- 3 Tables
  - SPO : subject, predicate, object
  - POS : predicate, object, subject
  - OSP : object, subject, predicate
- Store triples in the RowID of the table
- Take advantage of lexicographical sorting of row keys → fast range queries
- All patterns can be translated into a scan of one of these tables

Sample Triple Storage

Example RDF triple:

Subject	Predicate	Object
Alice	degreeFrom	Cornell

Stored RDF triple in Accumulo tables:

Table	Stored Triple
SPO	Alice, degreeFrom, Cornell
POS	degreeFrom, Cornell, Alice
OSP	Cornell, Alice, degreeFrom

Triple Patterns to Table Scans

Triple Pattern	Table to Scan
(Alice, degreeFrom, Cornell)	Any table (SPO default)
(Alice, degreeFrom, *)	SPO
(Alice, *, Cornell)	OSP
(*, degreeFrom, Cornell)	POS
(Alice, *, *)	SPO
(*, degreeFrom, *)	POS
(*, *, Cornell)	OSP
(*, *, *)	any full table scan (SPO default)

Query Processing

```
SELECT ?x WHERE {
        ?x takesCourse DBCourse .
        ?x rdf:type GraduateStudent . }
```

Step 1: POS – scan range

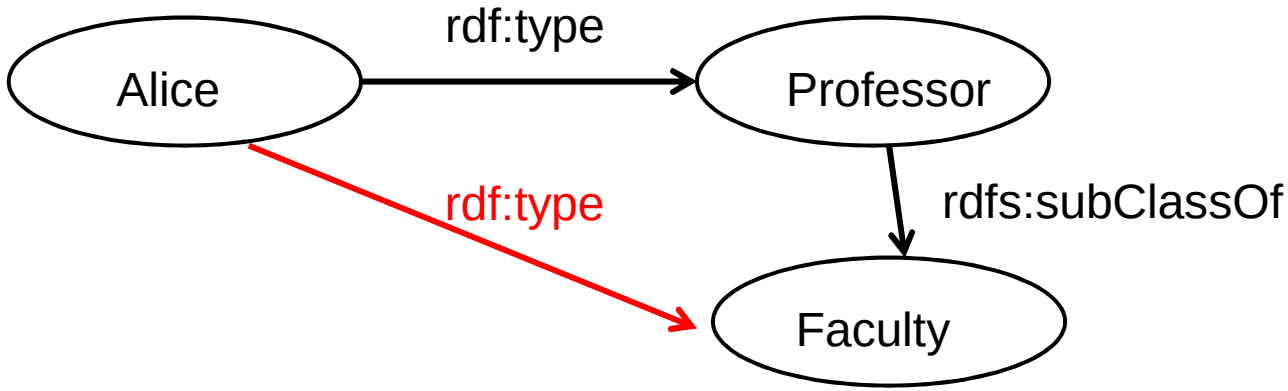
…
rdf:type, Professor, Alice
takesCourse, AICourse, John
takesCourse, AICourse, Zack
takesCourse, DBCourse, Bob
takesCourse, DBCourse, Greta
takesCourse, DBCourse, John
takesCourse, HCICourse, Alice
…

Step 2: for each ?x, SPO – index lookup

…
Bob, rdf:type, UndergradStudent
…
Greta, rdf:type, GraduateStudent
…
John, rdfType, GraduateStudent
…

Query Processing using Inference

SELECT ?x WHERE { ?x rdf:type Faculty }



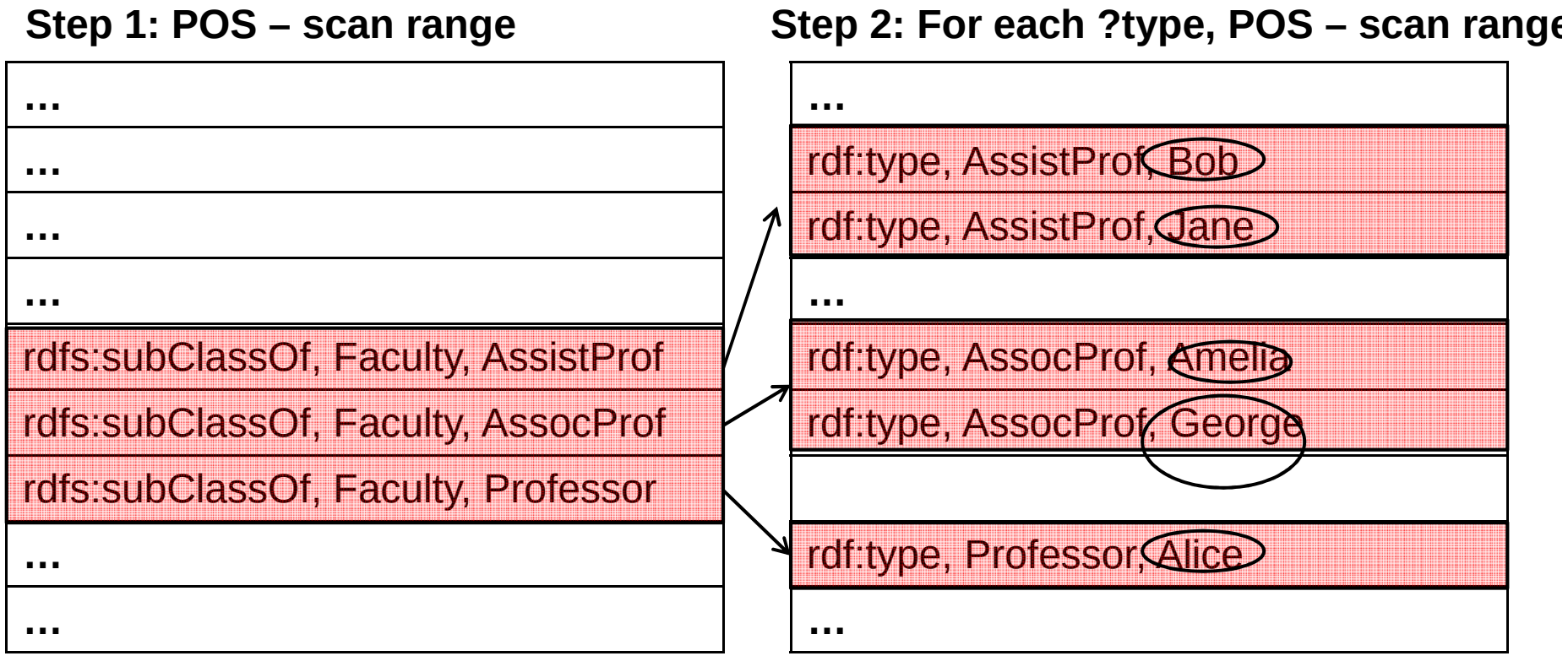
New query: SELECT ?x WHERE {

?type rdfs:subClassOf Faculty .

?x rdf:type ?type }

Query Plan for Expanded Query

```
SELECT ?x WHERE {
        ?type rdfs:subClassOf Faculty.
        ?x rdf:type ?type . }
```



Performance Enhancements

- Statistics Collection
- Parallel Joins
- Accumulo Batch Scanner use
  - Decreases network connections by up to 1K fold
- Time Ranges
  - Allow RDF querying on a small subset of data (based on a time loaded)

Optimized Joins with Statistics

- Collect statistics about data distribution
- Most selective triple evaluated first
- Ex:

Value	Role	Cardinality
rdf:type	Predicate	1mil
Student	Object	400K
takesCourse	Predicate	800K
DBCourse	Object	200

```
SELECT ?x WHERE {
 ?x takesCourse DBCourse .
 ?x rdf:type Student . }
```

Vs.

```
SELECT ?x WHERE {
 ?x rdf:type Student .
 ?x takesCourse DBCourse }
```

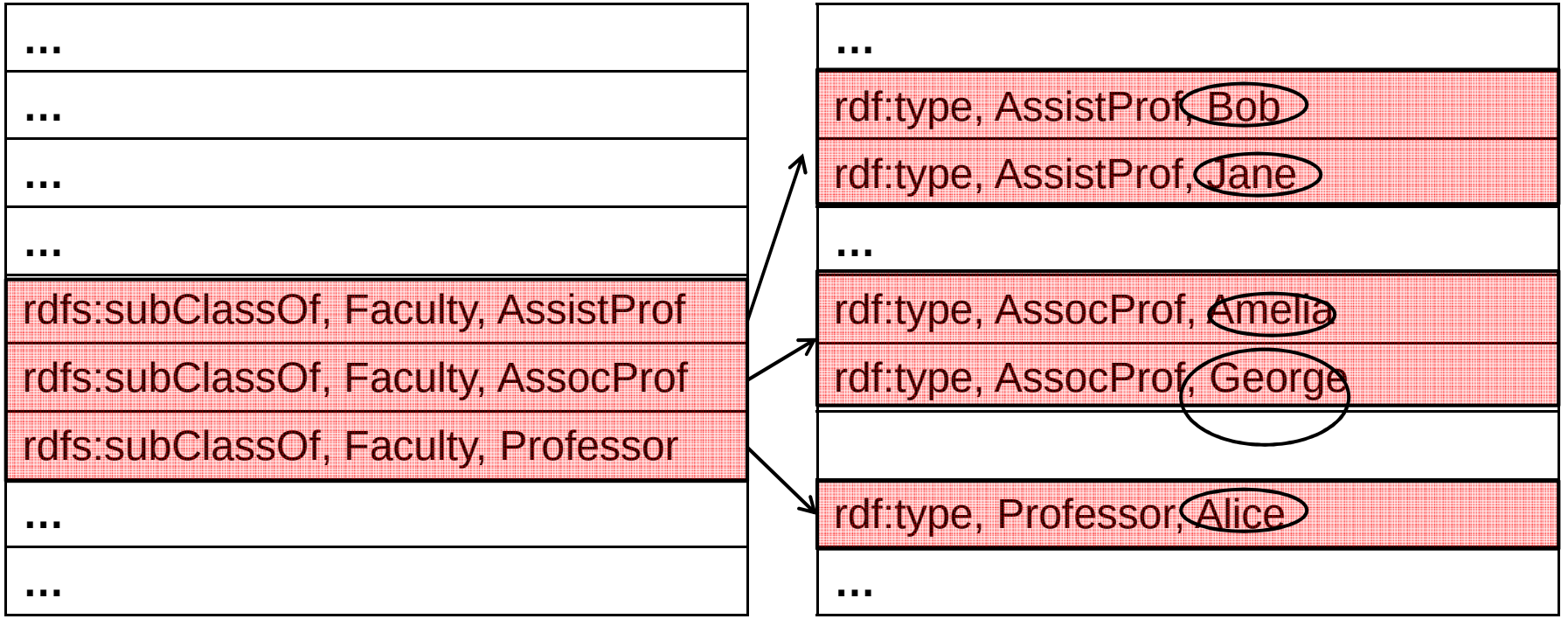
Enhancements * * * *

Parallel Joins

```
SELECT ?x WHERE {
        ?type rdfs:subClassOf Faculty.
        ?x rdf:type ?type . }
```

Step 1: POS – scan range

Step 2: For each ?type in parallel, POS – scan range



Enhancements * * * *

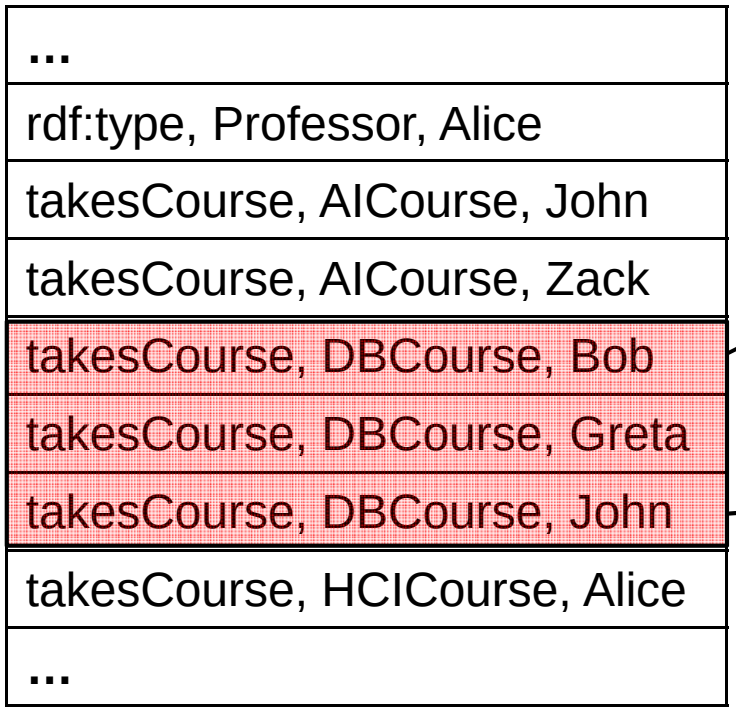
Batch Scanner

SELECT ?x WHERE {

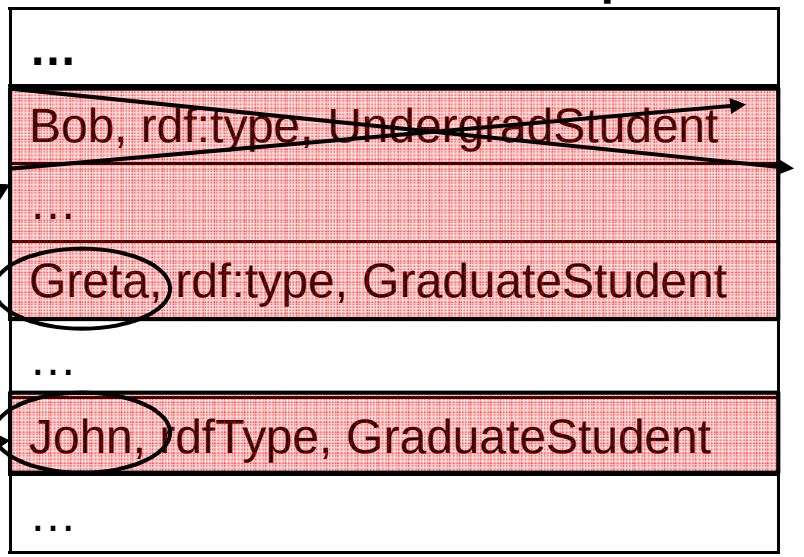
?x takesCourse DBCourse .

?x rdf:type GraduateStudent . }

Step 1: POS – scan range



Step 2: batched for each ?x, SPO – index lookup



Enhancements * * * *

Time Ranges

- SELECT ?load WHERE{
  ?measurement cpuLoad ?load .
  ?measurement timestamp ?ts .
  FILTER (?ts > “30 min ago”) }
- SELECT ?load WHERE{
  ?measurement cpuLoad ?load .
  ?measurement timestamp ?ts .
  timeRange (?ts,1300, 1330) }

Outline

- Problem
- Background
- Rya
  - Triple index
  - Performance enhancements
- Experimental results
- Conclusions and future work

Experiments Set-up

- Accumulo 1.3.0
  - 1 Accumulo master
  - 10 Accumulo tablet servers
- Each node: 8 core Intel Xeon CPU, 16 GB RAM, 3 TB Hard Drive
- Tomcat server for Rya
- Java implementation
- Dataset: LUBM

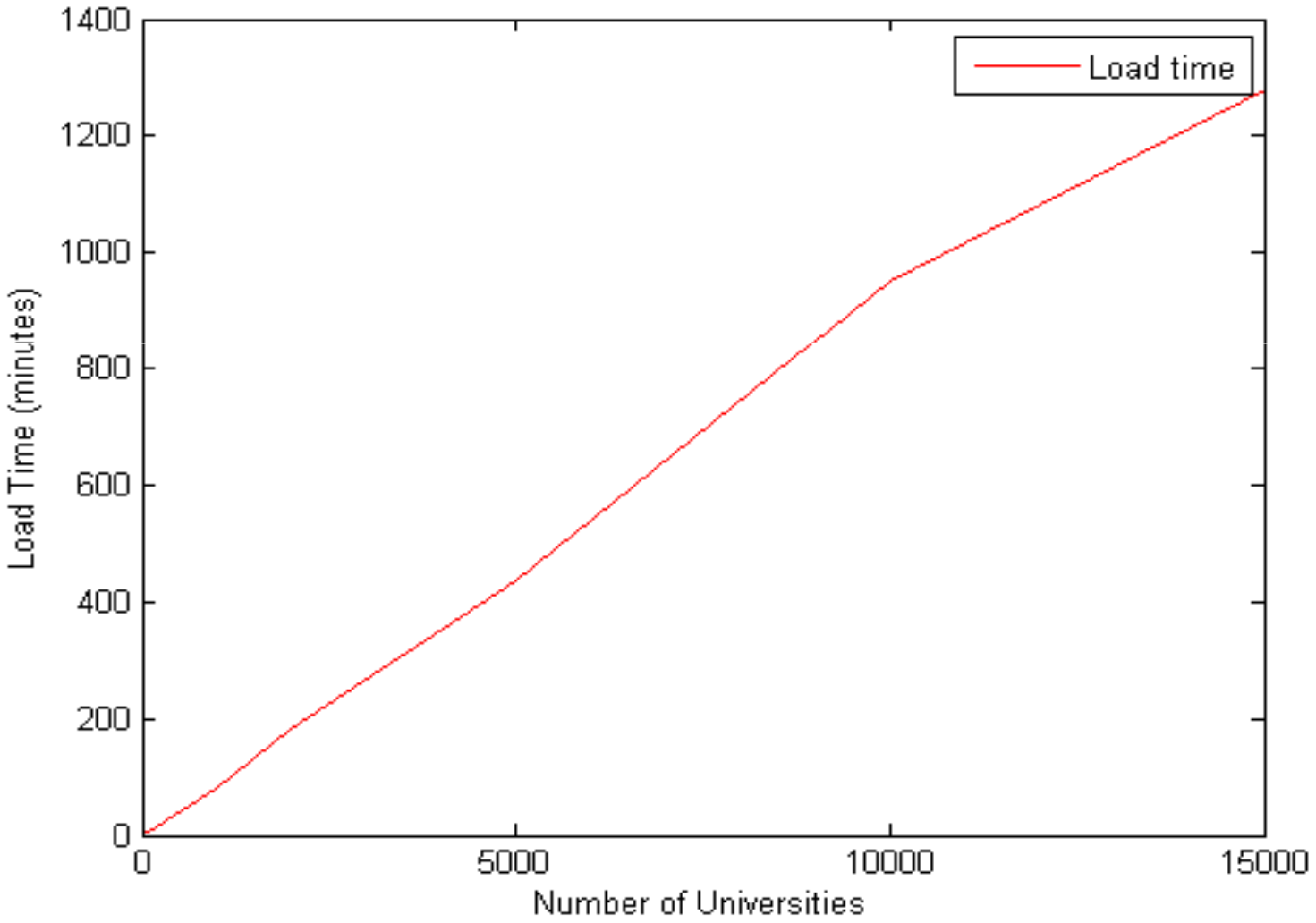
Performance Metrics

- LUBM data set – 10 to 15000 universities
- Load time
- Queries per second
  - Using batch scanner
  - Without batch scanner

Data Set - LUBM

Nb Universities	Nb Triples
10	1.3M
100	13.8M
1000	138.2M
2000	258.8M
5000	603.7M
10000	1.38B
15000	2.1B

Load time

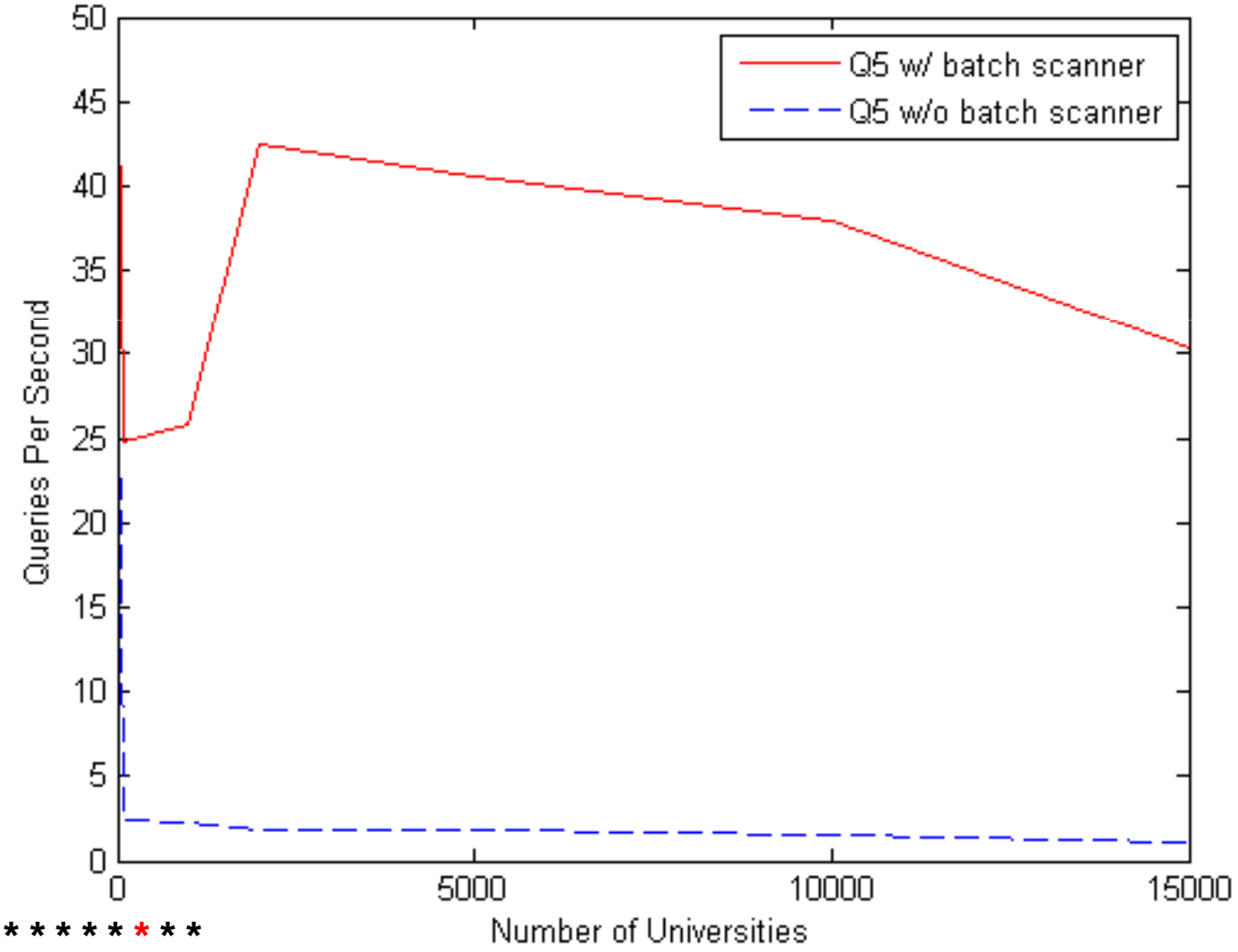


Experiments * * * * * * * *

Rya Query Performance - QpS

#Univ	10	100	1K	2K	5K	10K	15K
Q1	121.8	191.61	114.98	194.86	162.17	135.02	135.85
Q2	0.37	0.02	0.003	0.01	0.01	0.01	0.005
Q3	115.38	146.34	110.66	78.15	126.51	112.22	128.18
Q4	38.95	41.93	43.5	54.98	52.04	44.17	20.06
Q5	48.58	24.72	25.8	42.42	40.61	38.0	30.35
Q6	2.81	0.76	0.38	2.52	1.01	0.61	0.9
Q7	51.22	57.46	45.1	72.05	60.12	64.9	43.14
Q8	7.44	4.05	3.17	1.18	1.17	1.19	0.96
Q9	0.25	0.16	0.07	0.18	0.01	0.06	0.013
Q14	2.2	2.25	0.55	2.58	2.31	1.1	1.39

Query 5



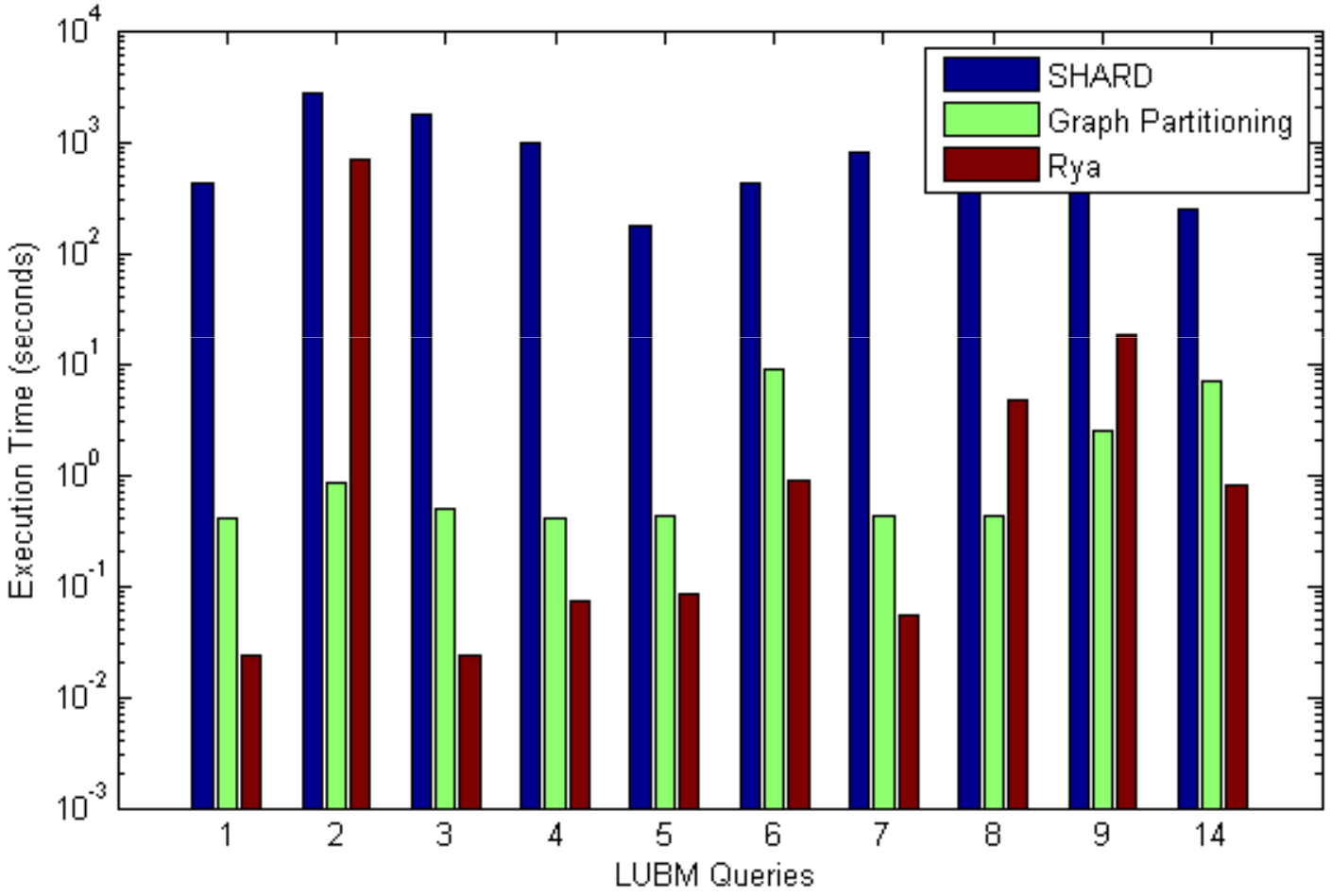
Experiments * * * * * * * * *

Comparison with Other Systems

- Systems:
  - Graph Partitioning [HAR11]
  - SHARD [RS10]
- Benchmark: LUBM 2000

System	Load Time
SHARD	10h
Graph Partitioning	4h 10min
Rya	3h 1min

Comparison with Other Systems



Experiments * * * * * * * *

Related Work

- RDF-3X [NW08] - centralized
- Graph Partitioning [HAR11] – graph partitioning + local RDF engines +MapReduce
- SHARD [RS10] – RDF triple store + HDFS
- Hexastore [WKB08] – six indexes
- SPARQL/MapReduce [MYL10] – MapReduce jobs to process SPARQL

Outline

- Problem
- Background
- Rya
  - Triple index
  - Performance enhancements
- Experimental results
- Conclusions and future work

Conclusions and Future Work

- Rya – scalable RDF Triple Store
  - Built on top of Accumulo and OpenRDF
  - Handles billions of triples
  - Millisecond query time for most queries
- Future:
  - Broader inferencing rules
  - New join algorithms

Thank You!

Questions?