Optimizing Sparql and Prolog for reasoning on large scale diverse ontologies

Jans Aasman, Ph.D.
CEO Franz Inc
Ja@Franz.com
This presentation

- Triples and a Graph database (2 minutes, I promise)
- AllegroGraph features
- Loading, indexing and querying: how did we do
- New numbers, 3.1 vs 3.2 and AllegroGraph vs (O)ther
- The secret sauce
createTripleStore("seminar.db")

addTriple (Person1 first-name Steve)
addTriple (Person1 isa Organizer)
addTriple (Person1 age 52)
addTriple (Person2 first-name Jans)
addTriple (Person2 isa Psychologist)
addTriple (Person2 age 50)
addTriple (Person3 first-name Craig)
addTriple (Person3 isa SalesPerson)
addTriple (Person3 age 32)

addTriple (Person1 colleague-of Person2)
addTriple (Person1 colleague-of Person3)

addTriple (Person1 likes Pizza)
addTriple ( Person3 neighbor-of Person1)
addTriple ( Person3 neighbor-of Person2)
And now you can query in Prolog or Sparql

```
(select (?xname ?yname)
  (q ?x colleague-of ?y)
  (q ?y neighbor-of ?x)
  (q ?x first-name ?xname)
  (q ?y first-name ?yname))

SELECT ?xname ?yname WHERE {
  ?y ex:first-name ?yname . }
```
Or reason

addTriple (first-name domain Person)

Every subject that has a predicate ‘first-name’ must be of type Person.
SPARQL Query

```
select ?x ?p ?o where {
}
```
### pAd-Track HA PGC-1 alpha

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability described by</td>
<td>Page?res?r=3&amp;stage=6&amp;vid=82&amp;tag=11&amp;filter=14427</td>
</tr>
<tr>
<td>Carcass sequence described by</td>
<td>19017</td>
</tr>
<tr>
<td>Is described in</td>
<td>16752370</td>
</tr>
<tr>
<td>Label</td>
<td>pAd-Track HA PGC-1 alpha</td>
</tr>
<tr>
<td>Sub Caco C1</td>
<td>Synthetic plasmid</td>
</tr>
</tbody>
</table>

AllegroGraph [1]

- Scalable and persistent Triple Store
  - Loads a 1.1 Billion triples in 20 hours on a single CPU and 8 hours on a 4 processor AMD machine (in federation)
- Federated
  - Create an abstract store that is a collection of other triple stores. Prolog and SPARQL and Reasoning work transparently against abstract store
- Compliant with standards
  - RDF, RDFS, OWL, SPARQL, Named Graphs, ISO Prolog, OWL-lite reasoning
- RDFS++ reasoner:
  - All of RDFS, inverseOf, sameAs, hasValue, transitiveProperty
- Full text indexing
- Java (Jena/Sesame) and Python interface.
AllegroGraph [2]

• Relational database efficiency for range queries
  – We support most XML schema types (dates, times, longitudes, latitudes, durations, telephone numbers, etc)

• Spatial database efficiency for geospatial primitives
  – Find elements in bounding boxes as fast as in spatial databases

• Temporal reasoning
  – Reasoning about times and intervals (Allen Logic)

• Social Network Analytics library
  – Find actor degrees and centrality, cliques, group centrality and cohesiveness
So how were we doing

- We were very fast at loading and indexing
- But queries on a reasoning store were slower than we wanted
Datasets we work with

- Science Commons (350,000,000 triples....)
- Linked Data (1,400,000,000 triples)
- LUBM8000 (1,200,000,000 triples)
Scientific Questions and Sources

“Find me genes involved in signal transduction that are related to pyramidal neurons!”
Find the socio-economic indicators for the place where Obama was born
SWAT Projects - the Lehigh University Benchmark (LUBM)

Overview
The Lehigh University Benchmark is developed to facilitate the evaluation of Semantic Web repositories in a standard and systematic way. The benchmark is intended to evaluate the performance of those repositories with respect to extensional queries over a large data set that commits to a single realistic ontology. It consists of a university domain ontology, customizable and repeatable synthetic data, a set of test queries, and several performance metrics.

References:
For an in-depth description of the benchmark and some evaluations done with its OWL version, refer to:
- [3] "An Evaluation of Knowledge Base Systems for Large OWL Datasets"

For some evaluations with the benchmark's DAML+OIL version, refer to:
- [2] "Choosing the Best Knowledge Base System for Large Semantic Web Applications"
- [1] "Benchmarking DAML+OIL Repositories"

Components
- **Ontology:**
  The benchmark ontology is named Univ-Bench. It has two versions: OWL Version and DAML Version.
- **Data Generator (UBA):**
  This tool generates synthetic OWL or DAML+OIL data over the Univ-Bench ontology in the unit of a university. These data...
A small part of the class hierarchy of LUBM
A small part of the property descriptions of LUBM

- Advisor
- Affiliated org
- AffiliateOf
- Age
- degreeFrom
- doctDegreeFrom
- emailAddress
- hasAlumnus

Domain
- Range
- InverseOf
- SubpropertyOf

Person
Professor
University
Organization
Our LUBM Benchmarks..

- Lubm 50 => 7,000,000 triples
- Lubm 8000 => 1,100,000,000 triples

- We use a 4 processor, 1.8 GHz, 16 Gig machine with 64 bit Fedora Core.

- We compare 3.1 against 3.2
- And (O)ther against 3.2
LuBM(50) Total query time

- AllegroGraph 3.1: 300 seconds
- AllegroGraph 3.2: 0 seconds

Series 1
AllegroGraph LUBM(50) Comparison

Seconds

Queries
LUBM(50) with medium queries zeroed

<table>
<thead>
<tr>
<th>Queries</th>
<th>AllegroGraph 3.1</th>
<th>AllegroGraph 3.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.02</td>
<td>0.02</td>
</tr>
<tr>
<td>3</td>
<td>0.06</td>
<td>0.06</td>
</tr>
<tr>
<td>7</td>
<td>0.04</td>
<td>0.04</td>
</tr>
<tr>
<td>9</td>
<td>0.08</td>
<td>0.08</td>
</tr>
<tr>
<td>11</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>13</td>
<td>0.1</td>
<td>0.1</td>
</tr>
</tbody>
</table>
So does this work for huge triplestores?

![LUBM(8000) Total query time diagram]

- **AllegroGraph 3.2**
- **Other**

Seconds

0 200 400 600 800 1000 1200

**LUBM(8000) Total query time**

**Total**

- Series1
LUBM(8000) with long queries zeroed

Queries

Seconds

AllegroGraph 3.2

Other
So what is the big deal? [1]

- AllegroGraph does not Materialize

- Typical triplestore:
  - Load & Index
  - Materialize: Do type inferences, some predicate normalizations
  - Index again

- With 3.2
  - Much more dynamic, add a few triples, delete or change an ontology
  - And back in the query business within a few minutes for a billion triples.
So what is the big deal? [2]

![Graph showing LUBM(8000) Total Time for Other/Static, AllegroGraph 3.2, and AllegroGraph 3.2 Federated. The graph includes total query time, type materializations, and loading and indexing times.]
So what is the big deal? [1]

- AllegroGraph does not Materialize

- Typical triplestore:
  - Load & Index
  - Materialize: Do type inferences, some predicate normalizations
  - Index again

- With 3.2
  - Much more dynamic, add a few triples, delete or change an ontology
  - And back in the query business within a few minutes for a billion triples.
So how do we do this?
# SPARQL - raw

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX u0d0: <http://www.Department0.University0.edu/>
PREFIX ub: <http://www.lubm.com/ontology#>

SELECT DISTINCT ?X ?Y ?Z WHERE
{ 
  ?Z rdf:type ub:Department .
  ?X ub:undergraduateDegreeFrom ?Y .
  ?X rdf:type ub:GraduateStudent .
  ?Y rdf:type ub:University . }
# SPARQL - cooked

(sparql.parser::sparql
 :where
 (#(?Z ![http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
   ![http://www.lubm.com/ontology#Department>))
 (#(?Z ![http://www.lubm.com/ontology#subOrganizationOf> ?Y)
 (#(?X ![http://www.lubm.com/ontology#undergraduateDegreeFrom> ?Y)
 (#(?X ![http://www.lubm.com/ontology#memberOf> ?Z)
 (#(?X ![http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
   ![http://www.lubm.com/ontology#GraduateStudent>))
 (#(?Y ![http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
   ![http://www.lubm.com/ontology#University>}})
# Prolog

(select0 (?x ?y ?z)
  (q- ?z !rdf:type !ub:Department)
  (q- ?z !ub:subOrganizationOf ?y)
  (q- ?x !ub:undergraduateDegreeFrom ?y)
  (q- ?x !ub:memberOf ?z)
  (q- ?x !rdf:type !ub:GraduateStudent)
  (q- ?y !rdf:type !ub:University)
)

A statistics based plan with some reasoning simplifications

# Planned

;; Return 3 variables: ?x, ?y, ?z
(db.agraph::find-or-create-map #:map13983 {advisor} :object :subject)
(db.agraph::find-or-create-map #:map13984 {takesCourse} :object :subject)
(db.agraph::find-or-create-map #:map13985 (?? +rdf-type-uri+) :object :subject)

;; estimate: 108,288 results per binding
(q/upis- ?y {teacherOf} ?z)

;; estimate: 9 results per binding
(db.agraph::q-upi-table #:map13983 ?y ?x)

;; estimate: 1 results per binding
(db.agraph::q-upi-table #:map13984 ?z ?x)

;; estimate: 4 results per binding
(lispp*
 (or (db.agraph.upi-maps:upi-pair-present-p #:map13985 {UndergraduateStudent} ?x)
 (db.agraph.upi-maps:upi-pair-present-p #:map13985 {GraduateStudent} ?x)
 (db.agraph.upi-maps:upi-pair-present-p #:map13985 {ResearchAssistant} ?x)
 (db.agraph.upi-maps:upi-pair-present-p #:map13985 {Student} ?x)))}
(select-internal (?x ?y ?z)
 (db.agraph::find-or-create-map #:map13983 {advisor} :object :subject)
 (db.agraph::find-or-create-map #:map13984 {takesCourse} :object :subject)
 (db.agraph::find-or-create-map #:map13985 (?? +rdf-type-uri+) :object :subject)
 !
 (q/upis- ?y {teacherOf} ?z)
 (db.agraph::q-upi-table #:map13983 ?y ?x)
 (db.agraph::q-upi-table #:map13984 ?z ?x)
 (lispp*
  (or (db.agraph.upi-maps:upi-pair-present-p #:map13985 {UndergraduateStudent} ?x)
     (db.agraph.upi-maps:upi-pair-present-p #:map13985 {GraduateStudent} ?x)
     (db.agraph.upi-maps:upi-pair-present-p #:map13985 {ResearchAssistant} ?x)
     (db.agraph.upi-maps:upi-pair-present-p #:map13985 {Student} ?x))))
Well, to be honest, really compiled down to machine instructions
Concluding with some reality

- Expect 3.2 in a few days. Call if you want prelease now.

- The prolog query optimizer will work for you

- The Sparql will still run on our old reasoner, expect the faster Sparql on our next release
Thank you