Object-UOBM

An Ontological Benchmark for Object-oriented Access

Martin Ledvinka
martin.ledvinka@fel.cvut.cz

Department of Cybernetics
Faculty of Electrical Engineering
Czech Technical University in Prague

September 29, 2015

Agenda

Why Another Ontology Benchmark?

There are ontology benchmarks. Do we need another one?

Berlin SPARQL Benchmark (BSBM)

Lehigh University Benchmark (LUBM)

University Ontology Benchmark (UOBM)

Why Another One?

BSBM, LUBM and UOBM are generic and not suitable for the *business application access* case.

Business Application Access

We consider domain tailored information systems, built using object-oriented paradigm, exploring and updating ontological sources. Generic semantic web and linked data applications are not considered.

Motivation

Our case – **business application access to ontologies**.

Approaches:

Domain-independent: Work on statement level (triple, axiom), no compiled information about the ontology schema. OWLAPI, Sesame API, Jena API.

Domain-specific: **Using object-ontological mapping to map ontology to objects. Empire, AliBaba, JOPA.**

Why OOM? To Turn This...

```java
private Map<String, Object> findPerson(URI pk, Map<URI, Map<String, Object>> knownPeople)
        throws RepositoryException {
    final Map<String, Object> values = new HashMap<>();
    RepositoryResult<Statement> r = connection.getStatements(pk, RDF.TYPE, null, false);
    final Set<String> types = new HashSet<>();
    boolean found = false;
    while (r.hasNext()) {
        final Statement s = r.next();
        if (s.getObject().stringValue().equals(personType)) {
            found = true;
        } else {
            types.add(s.getObject().stringValue());
        }
    }
    assert found;
    values.put("types", types);
    knownPeople.put(pk, values);
    r = connection.getStatements(pk, vf.createURI(firstName), null, false);
    Object value = getValue(r, Literal.class);
    values.put("firstName", value);
    ...
    final Set<Map<String, Object>> friends = new HashSet<>();
    r = connection.getStatements(pk, vf.createURI(friendOf), null, false);
    while (r.hasNext()) {
        final Statement s = r.next();
        if (!(s.getObject() instanceof URI)) {
            continue;
        }
        final URI friend = (URI) s.getObject();
        if (knownPeople.containsKey(friend)) {
            friends.add(knownPeople.get(friend));
        } else {
            friends.add(findPerson(friend, knownPeople));
        }
    }
    values.put("friends", friends);
    return values;
}

private Object getValue(RepositoryResult<Statement> values, Class<?> cls) throws RepositoryException {
    Object value = values.hasNext() ? values.next().getObject() : null;
    if (value != null && !cls.isAssignableFrom(value.getClass())) {
        throw new IllegalArgumentException();
    }
    return value;
}
```

Into This...

```
@OWLClass(iri="http://example.org/Student")
public class Student {
  @Id(generated = true)
  URI id;
  @DataProperty(iri="http://example.org/name")
  String name;
  @DataProperty(iri="http://example.org/email")
  String email;
  @ObjectProperty(iri="http://example.org/course")
  @ParticipationConstraints({
   @ParticipationConstraint(min=1,
     owlObjectIRI="http://example.org/course")
  })
  Set<Course> courses;
  @Inferred
  @Types
  Set<String> types;
  @Properties
  Map<String, Set<String>> properties;
}
```

And This

```java
public Student find(URI pk) {
    return em.find(Student.class, pk);
}
```

Benefits of OOM

- Cohesive domain objects,
- Objects can have behaviour,
- Less verbose, less error prone code,
- Easier to maintain,
- Enforcement of data structure valid for the application,
- Faster development.

OOM-based Benchmark

Business application access to ontologies has to provide:

- Create, Retrieve, Update, Delete (CRUD) operations,
- Access to inferred knowledge, classes, properties,
- Complex queries and meta-queries (SPARQL-DL),

CRUD in OOM

- Find individual and its properties,
- Fetch join – find individuals referenced by object properties of an individual,
- Insert individual and its properties,
- Delete individual's property value(s) and assert new one(s),
- Delete individual and its properties.

Queries p. I

Q_{S1}

```
SELECT DISTINCT ?x ?y WHERE {
  dept:Student119 ?x ?y.
}
```

- Select individual and all its property values,
- Get superset of data required by object *find* in OOM.

Queries p. II

Q_{S2}

```
SELECT ?name ?surname ?email ?course ?friend ?advisor
        ?degree ?neighbour ?type WHERE {
  {dept:Student119 benchmark:firstName ?name . }
  UNION
  {dept:Student119 benchmark:lastName ?surname . }
  UNION
  {dept:Student119 benchmark:emailAddress ?email . }
  UNION
  {dept:Student119 rdf:type ?type . }
  ...
}
```

- Select individual and a predefined set of its property values,
- Get exact set of data required by object *find* in OOM,
- Q_{S2OPT} – analogous to Q_{S2}, but using $\mathrm{OPTIONAL}$.

Queries p. III

Q_{S3}

```
SELECT ?x ?name ?author WHERE {
  ?x rdf:type benchmark:Publication ;
    benchmark:name ?name ;
    benchmark:publicationAuthor ?author.
}
```

- Select individuals of the given type + their properties,
- Not directly needed by OOM, but *findAll* is common in applications.

Queries p. IV

Q_{S4}

```
SELECT ?alumnus WHERE {
  <http://www.University0.edu>
    benchmark:hasAlumnus ?alumnus .
}
```

- Select value(s) of a property for the given individual,
- Attribute *lazy loading* support in OOM.

Queries p. V

Q_{U5}

```
DELETE {
  dept:Publication112 benchmark:name ?name .
}
INSERT {
  dept:Publication112 benchmark:name "Publication I" .
} WHERE {
  dept:Publication112 benchmark:name ?name . }
}
```

- Deletes property assertions and inserts new ones,
- Attribute *update* in OOM.

Queries p. VI

Q_{I6}

```
INSERT DATA {
  dept:Student117 a benchmark:Student ;
    a benchmark:SportsLover ;
    benchmark:firstName "John" ;
    benchmark:lastName "117" ;
    benchmark:emailAddress "John117@University0.edu" ;
    benchmark:isFriendOf
      dept:UndergraduateStudent123 ;
    benchmark:takesCourse
      dept:GraduateCourse12 .
}
```

- Insert individual's class and property assertions,
- Instance *persist* in OOM.

Queries p. VII

Q_{D7}

```
DELETE WHERE {
  dept:Student117 benchmark:firstName ?firstName . } ;
DELETE WHERE {
  dept:Student117 benchmark:lastName ?lastName . } ;
DELETE WHERE {
  dept:Student117 benchmark:emailAddress ?email . } ;
DELETE WHERE {
  dept:Student117 benchmark:isFriendOf ?friend . } ;
DELETE WHERE {
  dept:Student117 benchmark:takesCourse ?course .}
```

- Removes the specified assertions about an individual,
- Instance *remove* in OOM (*epistemic remove*).

Inference Strategies (Assumptions)

Materialization

+ Fast query answering,
– Slower statement insertion (and bulk loading),
– Slower statement retraction,
– Storage inflation,
– Inference expressiveness has to be specified on creation,
- GraphDB-SE 6.1 SP1.

Real-time Inference

– Slow query answering if inference is involved,
+ Faster insertion, data modification,
+ Inference level can be specified per query,
- Stardog 3.0.

Set Up

PC

- Linux Mint 17 64-bit
- CPU Intel i5 2.67 GHz (4 cores)
- 8 GB RAM
- 500 GB SATA HDD
- Java 8u40, -Xms6g -Xmx6g

Server

- Linux Debian 3.2.65 64-bit
- CPU Intel Xeon E3-1271 3.60 GHz (8 cores)
- 32 GB RAM
- 100 GB SSD HDD
- Java 8u40, -Xms20g -Xmx20g

Bulk Loading

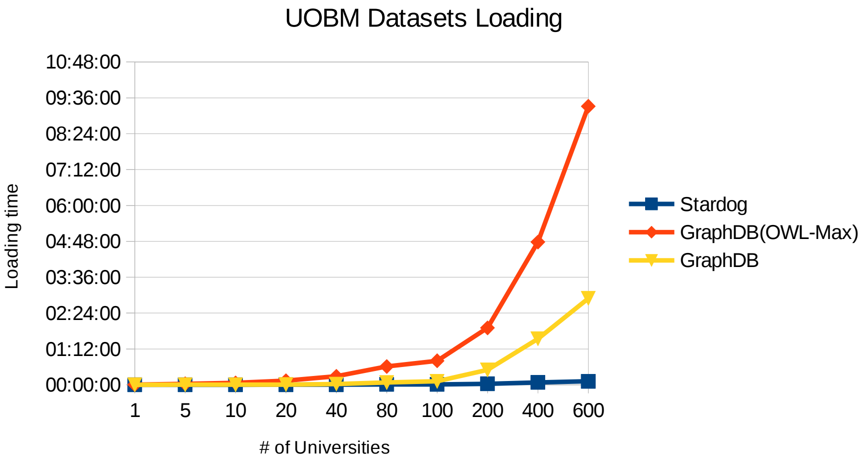
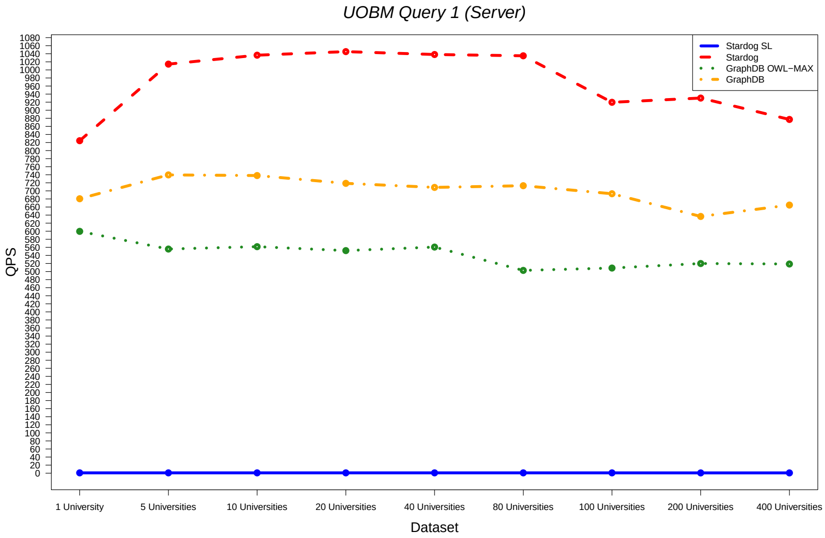
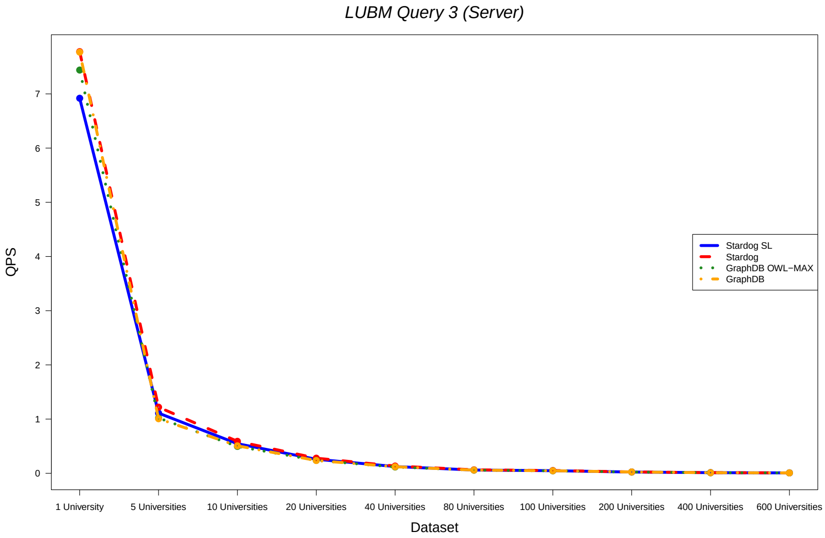


Figure : Dataset loading on server.

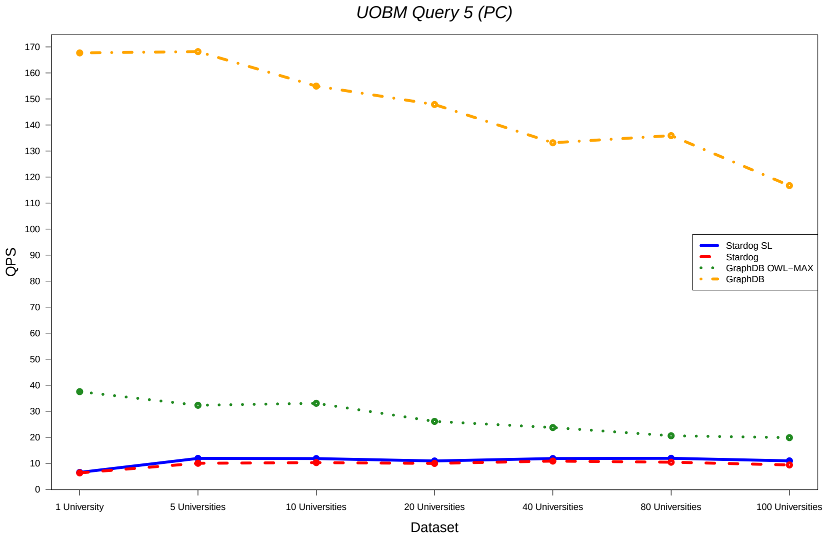
Q_{S1} on Server



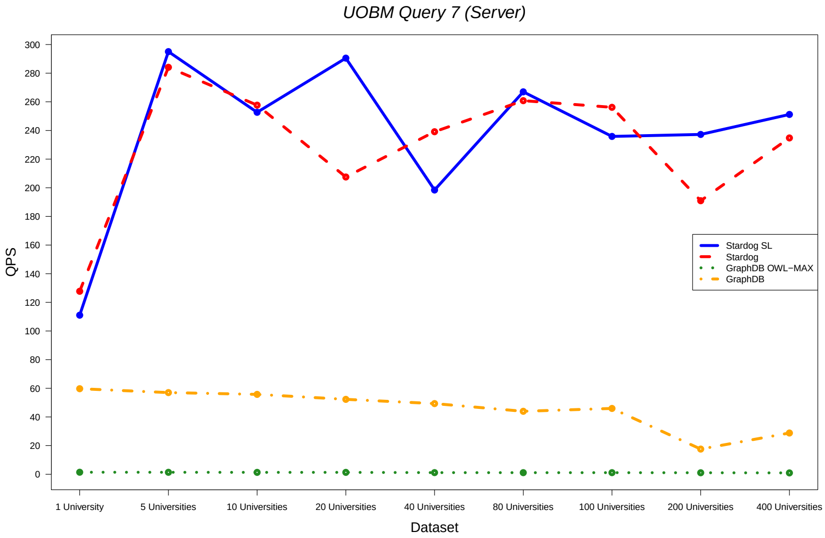
Q_{S3} on Server



Q_{U5} on PC


UOBM Query 5 (PC)
QPS
170
160
150
140
130
120
110
100
90
80
70
60
50
40
30
20
10
0
1 University
5 Universities
10 Universities
20 Universities
40 Universities
80 Universities
100 Universities
Dataset
Stardog SL
Stardog
GraphDB OWL−MAX
GraphDB

Q_{D7} on PC


UOBM Query 7 (Server)
QPS
0
20
40
60
80
100
120
140
160
180
200
220
240
260
280
300
1 University
5 Universities
10 Universities
20 Universities
40 Universities
80 Universities
100 Universities
200 Universities
400 Universities
Dataset
Stardog SL
Stardog
GraphDB OWL−MAX
GraphDB

Benchmark Summary

GraphDB	Stardog
• Fast for Q_{U5} and Q_{I6} • Slow for Q_{D7} with inference	• Fast bulk loading • Fast Q_{S1}, Q_{S2}, Q_{S3}, Q_{S4} w/o inference • Slow Q_{S1}, Q_{S2}, Q_{S3}, Q_{S4} with inference • Big fluctuations on server for Q_{U5}, Q_{I6} and Q_{D7}

- When large number of results (thousands+), performance drops significantly,
- Q_{S1} faster than Q_{S2}, $Q_{S2^{OPT}}$ performs much worse.

Conclusions

- Contemporary ontology benchmarks do not fit the application access scenario very well,
- OOM requires a rather small set of operations,
- Storages perform well when reading, updates are 3-4 times slower,
- Materialized knowledge can multiply the database size,
- Real-time reasoning is a significant performance issue,
- Materialized knowledge is not as big problem for updates.

The End

Thank you