Daniel Blum, Sara Cohen

GRR – Generating Random RDF

Extended Semantic Web Conference 2011

School of Computer Science and Engineering
The Hebrew University of Jerusalem
daniel.blum@mail.huji.ac.il
Motivation

- Testing Data-Intensive Applications is hard
  - Large volumes of data required
  - Expensive to create by hand
  - Not necessarily readily available
- Testing Semantic Web Data-Intensive Applications is even harder
  - Intricate format expected for input
- We focus on Semantic Web applications with well-defined data structure/distribution
The Goal

- To make testing easier by automatically generating data, when given
  - A schema
  - Data distributions
Presentation Outline

- Generating data example
- What’s available now?
- GRR – Generating Random RDF
- GRR Abstract & Concrete languages
  - Abstract Generation Language
  - Concrete Generation Language
- Optimization techniques
- Experimentation
- A glimpse to the system
- Conclusion
Generating Data Example

(C1) Create University

HUJI
(C2) Create 1-5 Department subOrg of University
(C3) Create 0-3 FullProfessor (FacultyMember) worksFor Department

Extended Semantic Web Conference (ESWC) 2011
(C4) one of the FullProfessors is headOf the Department

Generating Data Example
(C5) For each FacultyMember create 1-4 UnderGrad memberOf Department

- Grad1
  - Sam
    - sam@huji

- Grad2
  - Bob
    - bob@huji

- Grad3
  - Sally
    - sally@huji

- FullProf1
- FullProf2
- FullProf3

- CS
- Biology

- HUJI
(C6) 20% of Undergrad have a professor as Advisor.

Diagram:
- HUJI
  - CS
  - Biology
  - FullProf1
  - FullProf2
  - FullProf3
- Grad1
  - Grad2
  - Grad3
- Sally
  - bob@huji
  - Sam
  - sally@huji
  - sam@huji
What’s Available Now?

- **Downloadable RDF datasets**
  - Examples: Barton libraries, UniProt catalog, WordNet, LUBM, SP²Bench, Berlin SPARQL Benchmark
  - Good for testing efficiency of RDF storage systems

- **Data generators**
  - Example: SIMILE Project (RDFizers), RBench
  - Generate data according to user constraints
What’s Available Now?

- **Downloadable RDF datasets**
  - Examples: Barton libraries, UniProt catalog, WordNet, BSBM, SP²Bench, Berlin SPARQL Benchmark
  - Good for testing efficiency of RDF storage systems

- **Data generators**
  - Example: SIMILE Project (RDFizers), RBench
  - Generate data according to user constraints
Our System: GRR

- **GRR = Generating Random RDF**

- **Main Features**
  - Produces data with complex graph structure
  - Intuitive SPARQL like syntax
  - Draws the data values from desirable domains
  - Commands are translated into SPARQL queries applied directly to the RDF storage system
GRR: Abstract and Concrete Generation Languages
GRR: Abstract and Concrete Generation Languages

- The GRR system is built on an abstract logic-based language
  - Clearly defined expressive power
- Implemented as a multi-layer system
  - Different layers have varying levels of simplicity and expressiveness
Concrete Generation Language

- System overview
  - **Textual Interface** - Exposes intuitive way to generate the data
  - **RDF Interface** - A structured RDF based way to generate the data
  - **Java Interface** – The most flexible interface, but is implementation-oriented

- GRR – Textual Interface
- GRR – RDF Interface
- GRR – Java Interface
- ARQ-SPARQL
- Jena APIs
- TDB – RDF Storage
Abstract Generation Language
A data generation command:
- Maps/chooses existing parts of the graph by using queries
- Adds new nodes and/or new edges
- Iterates as needed
Abstract Generation Language

- Queries – Mapping the relevant nodes
  - Get Department subOrg University

Query maps Department
How do we choose numbers and values?

- Number of nodes to be created – *Number samplers* returning values from a given range and distribution
- Node Values and Attributes – *Value samplers* can return random or from a set with given values and distributions. For a newly generated node, we can add set of attributes, also with random values
Query results manipulation and usage

- A **Query-Sampler** defines **how many mappings will be returned**
  - Example: %20 of all FullProf nodes

- A **Query-Sampler** defines if a mapping can be returned more than once
  - Never return the same result
  - Return results that are distinct within the present inner queries
  - Allow all types of repetition
Composite Queries – Mapping different areas

- Connect GradStud with FullProf as advisor
  - Query 1 maps FullProf
  - Query 2 maps GradStud (must be connected to CS)

Add new ‘advisor’ edge
Concrete Generation Language
Concrete Generation Language

- GRR – Textual Interface
- GRR – RDF Interface
- GRR – Java Interface
- ARQ-SPARQL
- Jena APIs
- TDB – RDF Storage
Textual interface – Data Generation Template

- (FOR query sampling method
  
  [WITH (GLOBAL DISTINCT | LOCAL DISTINCT | REPEATABLE)]
  
  { list of classes }
  [WHERE { list of conditions } ])*

- [CREATE n-sampler { list of classes }]
- [CONNECT { list of connections }]

Extended Semantic Web Conference (ESWC) 2011
Generating Data Example

(C1) Create University

HUJI

(C1) CREATE 1 \{ub:Univ\}
(C2) Create 1-5 Department subOrg of University

(C2) FOR EACH {ub:Univ} CREATE 1-5 {ub:Dept} CONNECT {ub:Dept ub:subOrg ub:Univ}
(C3) Create 0-3 FullProfessor (FacultyMember) worksFor Department

(C3) FOR EACH { ub:Dept } CREATE 0-3 { ub:FullProf } 
CONNECT { ub:FullProf ub:worksFor ub:Dept }
(C4) one of the FullProfessors is headOf of the Department

(C4) FOR EACH { ub:Dept } FOR 1 { ub:FullProf }
WHERE { ub:FullProf ub:worksFor ub:Dept }
CONNECT { ub:FullProf ub:headOf ub:Dept }
(C5) For each FacultyMember create 1-4 UnderGrad memberOf Department

- FullProf1
- FullProf2
- FullProf3
- CS
- Biology
- HUJI

- Grad1
- Grad2
- Grad3
- Sam

- Sally
- Bob
- Sam
- Sally
- Bob
(C5) For each FacultyMember create 1-4 UnderGrad memberOf Department

(C5) FOR EACH {ub:Faculty, ub:Dept} WHERE {ub:Faculty ub:worksFor ub:Dept} CREATE 1-4 {ub:Undergrad} CONNECT {ub:Undergrad ub:memberOf ub:Dept}
(C6) 20% of Undergrad have a professor as Advisor

- Grad1
  - Bob
    - bob@huji
  - Sally
    - sally@huji
  - Grad2
  - Grad3

- FullProf1
- FullProf2
- FullProf3

- CS
- Biology
- HUJI

- Sam
  - sam@huji
(C6) 20% of Undergrad have a professor as Advisor

(C6) FOR 20%-20% { ub:Undergrad, ub:Dept } WHERE { ub:Undergrad ub:memberOf ub:Dept } FOR 1 with repeatable repetitions { ub:Prof } WHERE { ub:Prof ub:worksFor ub:Dept } CONNECT { ub:Undergrad ub:advisor ub:Prof }
Textual interface – FOAF Example

- namespace foaf: <http://xmlns.com/foaf/0.1/>
- CREATE 250000 { foaf:Person }
- FOR EACH { foaf:Person ?p1 }
- FOR 15-25 { foaf:Person ?p2 }
- WHERE { FILTER( ?p1 != ?p2 ) }
- CONNECT { ?p1 foaf:knows ?p2 }
Optimization Techniques
Caching Query Results

- We are using composite queries that might be called several times
- The same query can be executed several times therefore we cache query results
Avoiding Unnecessary Caching (‘Smart Cache’)

- Caching reduces the number of times a query will be applied, but it incurs a significant storage overhead
- In GRR, we avoid caching of results when caching is guaranteed to be useless
Experimentation
Experimention

- Setup
  - GRR uses Jena’s TDB database (pure Java non-SQL storage system)
  - Desktop running Windows Vista x64 with 4GB RAM (2GB allocated for the Java runtime heap)
Experimentation

- Recreating the LUBM benchmark
  - Time / Queries measurements

![Graphs showing time and queries for departments with different cache strategies.]
Experimention

- Recreating the LUBM benchmark
  - Cache size measurements

---

![Bar Chart: Cache (KB) for Departments]

- X-axis: Number of Departments (10, 20, 30, 40)
- Y-axis: Cache Size (KB)
- Two categories: Always Cache, Smart Cache

Extended Semantic Web Conference (ESWC) 2011
Experimentation

- GRR scalability while using FOAF schema
  - Time / Queries measurements

![Graphs showing time and queries for different numbers of people and projects with cache options.](image)
Experimentation

- GRR scalability while using FOAF schema
  - Cache size measurements

![Diagram showing cache size measurements for N People and M Projects.]
System Implementation - GUI

The random RDF output file was loaded to the screen successfully.
Conclusion
Conclusion

- GRR – System for generating random RDF data
- Useful for generating test data for Semantic Web applications
- Unique since it can create arbitrary structures
- Presents a method to create data that is both natural and powerful
Learn more / Contact us

- www.cs.huji.ac.il/~danieb12
- daniel.blum@mail.huji.ac.il
- GRR in GITHUB: Repository- danieb12/GRR
Questions?

Thanks for listening