Spatio-Temporal Gridded Data Processing on the Semantic Web

Andrej Andrejev, Dimitar Misev*, Peter Baumann*, Tore Risch

Department of Information Technology, Uppsala University
* Computer Science & Electrical Engineering Department, Jacobs University, Bremen

andrej.andrejev@it.uu.se
They never come alone...
Introduction

Big Data..
Introduction

Big Data..

Namely: Massive Numeric Datasets
  • typically ordered along a number of orthogonal axes
Big Data..

Namely: Massive Numeric Datasets
  • typically ordered along a number of orthogonal axes
They come together...
... with Metadata
Introduction

... with Metadata
Introduction

- axis names
- origins
- low and high bounds
...
Introduction

- field descriptions
- accuracy and noise
- reserved values
...

Andrej Andrejev, DSDIS Conference - December 2015, Sydney
Introduction
Introduction

- Domain
- Range
- Master instance
- Provenance

- input parameters
- tools & methods used

...
Introduction

They stay together...
They stay together...

...since our data model is RDF with Arrays
RDF with Arrays databases

• RDF is very suitable for describing properties about scientific experiments (metadata) **but:**
  • Arrays are represented in a very inefficient way in RDF

• **Our approach:** Extend RDF with compact numerical array representation
Scientific SPARQL

• SPARQL is very suitable for searching scientific RDF-based metadata, **but:**
  • SPARQL has no support for queries involving array operations

• **Our approach:** Extent SPARQL with common array operators => SciSPARQL
Reusing program libraries

• Often need for using existing program libraries when processing experiments data, **but**:
  • SPARQL has no standard way of plugging in external program libraries and algorithms

• **Our approach**: SciSPARQL provides a general mechanism to call functions in C, Java, Python, or Matlab
Our System Architecture

SciSPARQL queries / updates

SciSPARQL results

SSDM
SciSPARQL Database Manager

In-memory database

RDF importer

generic storage backend(wrapper interface

RasQL wrapper

Python, Java, MATLAB, .. engines

External functions

Andrej Andrejev, DSDIS Conference - December 2015, Sydney
Our System Architecture

SciSPARQL queries / updates

SciSPARQL results

SSDM
SciSPARQL Database Manager

In-memory database

RDF importer

generic storage backend/wrapper interface

Retrieving array data

Python, Java, MATLAB, .. engines

External functions

Andrej Andrejev, DSDIS Conference - December 2015, Sydney
Our System Architecture

SciSPARQL queries / updates

SciSPARQL results

SSDM
SciSPARQL Database Manager

In-memory database

RDF importer

Retrieving array data

generic storage backend/wrapper interface

RasQL wrapper

External functions

Python, Java, MATLAB, .. engines

functional extensibility

Delegating common array computations

Andrej Andrejev, DSDIS Conference - December 2015, Sydney
Introducing Geo Coverages
A Coverage ...
A Coverage ...

... is an abstract data structure that represents some space/time varying phenomenon
Coverage subtypes are available for regular and irregular grids, point clouds and meshes.
Coverages

A Coverage ...

... is an abstract data structure that represents some space/time varying phenomenon

According to ISO 19123 and its implementation model by Open Geospatial Consortium:

Coverage subtypes are available for regular and irregular grids, point clouds and meshes
A Grid Coverage
Grid Coverages

A Grid Coverage is identified by

- **coverage id**

- **domain set**, specifying spatio-temporal region of interest
  - **geometric grid**: integer coordinates
  - **rectified grid**: grid origin and offset vector in some CRS
  - **referenceable grid**: CRS coordinates for each position

- **range type**: structural description and technical metadata, required of appropriate understanding of a coverage
A Grid Coverage

is identified by

• coverage id

• domain set, specifying spatio-temporal region of interest
  – geometric grid: integer coordinates
  – rectified grid: grid origin and offset vector in some CRS
  – referenceable grid: CRS coordinates for each position

• range type: structural description and technical metadata, required of appropriate understanding of a coverage

A Range Type

is composed of one or more fields, each having its own:

• name identifier
• human-readable description
• type definition
• unit of measure
• list of reserved values
...
Grid Coverages

Grid Coverage Ontology

GridCoverage

- RangeType
  - NILValue
    - value
    - reason
  - Field
    - name
    - description
    - definition
    - isOptional
    - isUpdatable
  - Range
    - IntervalRange
    - TypedRange
      - EnumeratedRange
        - low
        - high
        - values
        - dataType
- RectifiedGridDomain
  - CRS
  - origin
  - offset
- ReferenceableGridDomain
  - axisNames
  - dimensionality
  - id
  - low
  - high
- coordinateGrid
Grid Coverages

Grid Coverage Ontology

GridDomain
- axisNames
- dimensionality
- low
- high
- CRS
- origin
- ReferenceableGridDomain
  - offset
- coordinateGrid
- RectifiedGridDomain

GridCoverage
- id
- RangeType
- NILValue
- Field
  - description
  - definition
  - isOptional
  - isUpdatable
  - values
    - dataType
    - low
    - high
  - IntervalRange
    - low
    - high
  - TypedRange
    - values
  - EnumeratedRange

Flexibility of RDF

Andrej Andrejev, DSDIS Conference - December 2015, Sydney
SciSPARQL query example
Normalized Difference Vegetation Index (NDVI)

Define function NDVI(?nir ?red ?x) as

\[
SELECT (255 * \text{xsd:integer}(((?nir - ?red) / (?nir + ?red)) > ?x)) \text{ AS ?result)}
\]

Create 0.5-threshold NDVI map of the coverage identified as "mycoverage", using its :nir and :red properties

\[
SELECT (\text{MAP(xsd:integer, NDVI(*, *, 0.5), ?nir_array, ?red_array}) \text{ AS ?result)}
\]

WHERE { ?c :id "mycoverage" ;
:nir ?nir_array ;
:red ?red_array }
Normalized Difference Vegetation Index (NDVI)

```
DEFINE FUNCTION NDVI(?nir ?red ?x) AS
```

Create 0.5-threshold NDVI map of the coverage identified as "mycoverage", using its :nir and :red properties

```
SELECT (MAP(xsd:integer, NDVI(*, *, 0.5), ?nir_array, ?red_array) AS ?result)
WHERE { ?c :id "mycoverage" ;
    :nir ?nir_array ;
    :red ?red_array }
```
DEFINE FUNCTION NDVI(?nir ?red ?x) AS

Create 0.5-threshold NDVI map of the coverage identified as "mycoverage", using its :nir and :red properties
SELECT (MAP(xsd:integer, NDVI(*, *, 0.5), ?nir_array, ?red_array) AS ?result)
WHERE { ?c :id "mycoverage" ;
    :nir ?nir_array ;
    :red ?red_array }
SciSPARQL Example

Normalized Difference Vegetation Index (NDVI)

```
DEFINE FUNCTION NDVI(?nir ?red ?x) AS
AS ?result) either 0 or 255

Create 0.5-threshold NDVI map of the coverage identified as "mycoverage", using its :nir and :red properties

SELECT (MAP(xsd:integer, NDVI(*, *, 0.5), ?nir_array, ?red_array)
AS ?result)
WHERE { ?c :id "mycoverage" ;
  :nir ?nir_array ;
  :red ?red_array }
```
SciSPARQL Example

Normalized Difference Vegetation Index (NDVI)

```
DEFINE FUNCTION NDVI(?nir ?red ?x) AS
```

Create 0.5-threshold NDVI map of the coverage identified as "mycoverage", using its :nir and :red properties

```
SELECT (MAP(xsd:integer, NDVI(*, *, 0.5), ?nir_array, ?red_array) AS ?result)
WHERE { ?c :id "mycoverage" ;
  :nir ?nir_array ;
  :red ?red_array }
```
SciSPARQL Example

Normalized Difference Vegetation Index (NDVI)

DEFINE FUNCTION NDVI(?nir ?red ?x) AS
AS ?result)

Create 0.5-threshold NDVI map of the coverage identified as "mycoverage", using its :nir and :red properties

SELECT (MAP(xsd:integer, NDVI(*, *, 0.5), ?nir_array, ?red_array)
AS ?result)
WHERE { ?c :id "mycoverage" ;
:nir ?nir_array ;
:red ?red_array }
SciSPARQL Example

Normalized Difference Vegetation Index (NDVI)

**SciSPARQL Example**

**Normalized Difference Vegetation Index (NDVI)**

Define function `NDVI` as:

```sparql
DEFINE FUNCTION NDVI(?nir ?red ?x) AS
```

Create 0.5-threshold NDVI map of the coverage identified as "mycoverage", using its :nir and :red properties:

```sparql
SELECT (MAP(xsd:integer, NDVI(*, *, 0.5), ?nir_array, ?red_array)) AS ?result
WHERE { ?c :id "mycoverage" ;
  :nir ?nir_array ;
  :red ?red_array }
```

Lexical closure:

- `NDVI(*)`: functions that receive a list of parameters are `lexical closure`
- `MAP(xsd:integer, NDVI(*, *, 0.5), ?nir_array, ?red_array)`: `map` functions that take lexical closure arguments are `lexical closure`

'Free' arguments - bound by the mapper:

- `?c :id "mycoverage"`
- `?nir_array` and `?red_array`
SciSPARQL Example

Normalized Difference Vegetation Index (NDVI)

\[
\text{DEFINE FUNCTION NDVI(} \text{nir, red, x) AS}
\]

\[
\text{SELECT}\ (255 \ast \text{xsd:integer}(((\text{nir} - \text{red}) / (\text{nir} + \text{red})) > \text{x}) \text{ AS result}) \rightarrow \text{either 0 or 255}
\]

Create 0.5-threshold NDVI map of the coverage identified as "mycoverage", using its :nir and :red properties

\[
\text{SELECT (MAP(xsd:integer, NDVI(*, *, 0.5), \text{nir_array, red_array}) AS result) }
\]

WHERE \{ \text{c :id "mycoverage" ;}
\text{ :nir \text{nir_array ;}
\text{ :red \text{red_array} }
\}

"mycoverage"

: id

?c : id "mycoverage"

: nir

?nir_array

: red

?red_array

MAP

NDVI(*, *, 0.5)

?result

Andrej Andrejev, DSDIS Conference - December 2015, Sydney
SciSPARQL Example

Normalized Difference Vegetation Index (NDVI)

```sparql
DEFINE FUNCTION NDVI(?nir ?red ?x) AS
AS ?result) -> either 0 or 255 :integer
```

Create 0.5-threshold NDVI map of the coverage identified as "mycoverage", using its :nir and :red properties

```sparql
SELECT (MAP(xsd:integer, NDVI(*, *, 0.5), ?nir_array, ?red_array)
AS ?result)
WHERE { ?c :id "mycoverage" ;
  :nir ?nir_array ;
  :red ?red_array }
```

'free' arguments - bound by the mapper

`lexical closure`

Andrej Andrejev, DSDIS Conference - December 2015, Sydney
SciSPARQL Example

Normalized Difference Vegetation Index (NDVI)

DEFINE FUNCTION NDVI(?nir ?red ?x) AS

Create 0.5-threshold NDVI map of the coverage identified as "mycoverage", using its :nir and :red properties, and insert it as :ndviMap property

INSERT { ?c :ndviMap ?result }
WHERE { ?c :id "mycoverage" ;
 :nir ?nir_array ;
 :red ?red_array .
BIND (MAP(xsd:integer, NDVI(*, *, 0.5),
 ?nir_array, ?red_array) AS ?result) }
Comparison

Doing the same thing
in a programming language of your choice?
Comparison

Normalized Difference Vegetation Index (NDVI)

DEFINE FUNCTION NDVI(?nir ?red ?x) AS
AS ?result)

Create 0.5-threshold NDVI map of the coverage identified as "mycoverage", using its \texttt{:nir} and \texttt{:red} properties

Find the right instance of a Gridded Coverage and all the associated data needed for the task

Andrej Andrejev, DSDIS Conference - December 2015, Sydney
Create 0.5-threshold NDVI map of the coverage identified as "mycoverage", using its :nir and :red properties.
Create 0.5-threshold NDVI map of the coverage identified as "mycoverage", using its :nir and :red properties

Find the right instance of a Gridded Coverage and all the associated data needed for the task

\[
\text{NDVI}(\text{nir}, \text{red}, x) \text{ AS}
\]
\[
\text{SELECT } (255 * \text{xsd:integer}(((\text{nir} - \text{red}) / (\text{nir} + \text{red})) > x) \text{ AS result})
\]

```sql
DEFINE FUNCTION NDVI(?nir ?red ?x) AS
```
Normalized Difference Vegetation Index (NDVI)

**DEFINE FUNCTION NDVI(?nir ?red ?x) AS**


Create 0.5-threshold NDVI map of the coverage identified as "mycoverage", using its :nir and :red properties

- Find the right instance of a Gridded Coverage and all the associated data needed for the task
- Hardcode a nested loop across all array elements
- Load
- Load & validate
- Allocate

NDVI(*, *, 0.5)
Benefits of combining data and metadata within a single query:

- More transparent and self-contained queries are easier to maintain.
- Result selection and postprocessing on the server whenever metadata is needed, it is in the same place.
- Single round-trip to the server instead of retrieving metadata in order to guide the data retrieval and processing.
- More freedom for the optimizer one complex query is better than a series of simple ones.
Summary of contributions

• **RDF with Arrays**
a flexible way to model Gridded Coverages
with any application-relevant metadata

• **Hybrid data store**
  storing RDF graphs in-memory and
  gridded data on disk (distributed and parallel)

• **SciSPARQL queries**
  combining graph patterns and array operations:
  metadata conditions and data filtering / postprocessing

• **Mediator architecture**
  performing computations where the data are
The software, documentation, and examples are available at

http://user.it.uu.se/~udbl/SciSPARQL

This work is supported by