RDSZ

An approach for lossless RDF stream compression

Norberto Fernández, Jesús Arias, Luis Sánchez, Damaris Fuentes
Dpto. Ingeniería Telemática, Universidad Carlos III de Madrid
Óscar Corcho
Ontology Engineering Group, Universidad Politécnica de Madrid

#eswc2014Garcia

Outline

- Motivation
- RDSZ algorithm
  - Introduction (inspiring principles)
  - Running example (working details)
- Evaluation
  - Datasets & setup
  - Compression performance
  - Response time
- Conclusions & Future lines

Motivation

- Popularization of streaming data on the web …
  - Social networks (e.g. micro-blogging)
  - Internet of things (e.g. sensor data)

Source: http://channelnomics.com/

Motivation

- Initiatives from the semantic web community …
  - RDF Stream Processing Group (W3C)
  - Stream reasoning
  - Query languages
    - SPARQLStream, C-SPARQL, …
  - Scalable RDF stream processing
    - CQELS-Cloud
  - Scalable RDF stream publishing
    - Ztreamy

Motivation

- Ztreamy
  - Middleware for scalable RDF stream publishing
    - Publish RDF streams on top of HTTP
    - Up to 40.000 clients per server with delays of few seconds
  - More info …

Ztreamy: A middleware for publishing semantic streams on the Web. J. Arias, N. Fernández, L. Sánchez, D. Fuentes. Journal of Web Semantics 25(0), 2014.

  - … or visit the demo at ESWC 2014

Motivation

- Ztreamy compresses the RDF streams
  - No RDF specific stream compressor available…
    - Only RDF block compressors (like HDT)
  - … use a general purpose stream compressor: Zlib (Deflate)

- We found that this **compression has a big impact on system performance**
  - Drastic reduction in network load
  - Reduction on CPU usage

http://beforeitsnews.com/

RDSZ: Introduction

- Research quest:

  Can we improve Zlib compression rate?

- Hypothesis: Yes, we can!
  - Zlib provides good compression rate for text ...
  - … items in an RDF stream have structural similarities …
    - Generated by software according to a schema
  - … and Zlib takes limited advantage of these similarities

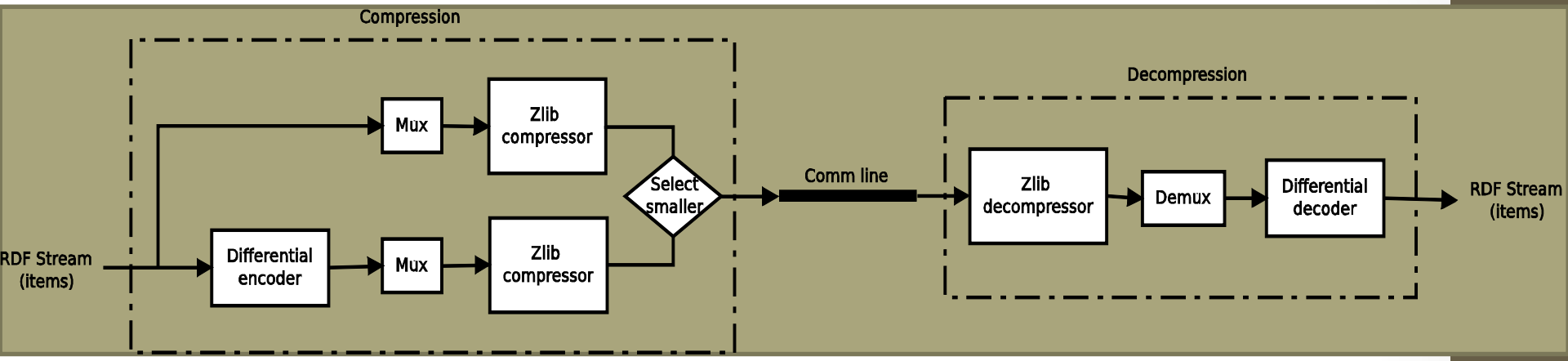
RDSZ: Introduction

- Idea!
  - Use **differential encoder** to exploit structural similarities …
    - Represent an RDF item on the basis of previous items in the same stream
      - Use a LRU cache to save information about previous items
    - The results are serialized as text
  - … combined **with Zlib**
    - Good text compression

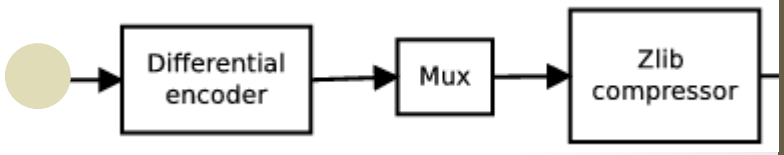
http://www.zlib.net/

RDSZ: Algorithm

- Block diagram …



- … I will describe the functionality of the most important blocks using a running example
  - See the paper for the algorithmic nuts & bolts



RDSZ: Algorithm

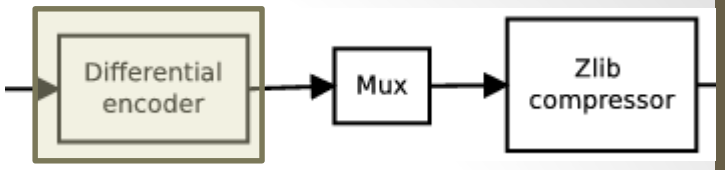
- Our sample stream has two items…
  - Item 1 (no previous item before this)

```
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix wtl: <http://webtlab.it.uc3m.es/> .

wtl:_556103084 dc:date "2013-02-20T16:58:32Z";
    dc:author "Wonderboy";
    wtl:pageid 6227038;
    wtl:title "Villeroy & Boch" .
```

  - Item 2 (sent after item 1)

```
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix wtl: <http://webtlab.it.uc3m.es/> .

wtl:_556103110 dc:date "2013-02-20T16:58:40Z";
    dc:author "Wonderboy";
    wtl:pageid 31317733;
    wtl:title "2013 Women's Cricket World Cup" .
```

RDSZ: Algorithm



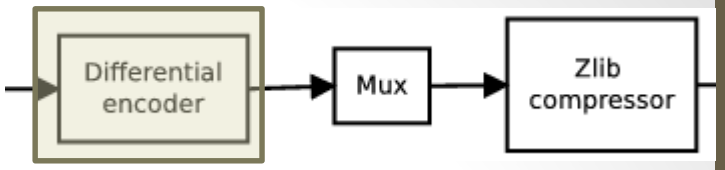
- Processing item 1 at the differential encoder
  1. Decompose the RDF item into **triple pattern** & **variable bindings**

```
?x0 <http://purl.org/dc/elements/1.1/author> ?x1 .
?x0 <http://purl.org/dc/elements/1.1/date> ?x2 .
?x0 <http://webtlab.it.uc3m.es/pageid> ?x3 .
?x0 <http://webtlab.it.uc3m.es/title> ?x4 .
```

variable	value
?x0	<http://webtlab.it.uc3m.es/_556103084>
?x1	"Wonderboy"
?x2	"2013-02-20T16:58:32Z"
?x3	6227038
?x4	"Villeroy & Boch"

  2. Is the triple pattern included in the encoder cache?
     - NO → append the Turtle serialization of item 1 to output
  3. Save the pattern and bindings in the encoder LRU cache for future reference

RDSZ: Algorithm



- Processing item 2 at the differential encoder
  1. Decompose the RDF item into **triple pattern** & **variable bindings**

```
?x0 <http://purl.org/dc/elements/1.1/author> ?x1 .
?x0 <http://purl.org/dc/elements/1.1/date> ?x2 .
?x0 <http://webtlab.it.uc3m.es/pageid> ?x3 .
?x0 <http://webtlab.it.uc3m.es/title> ?x4 .
```

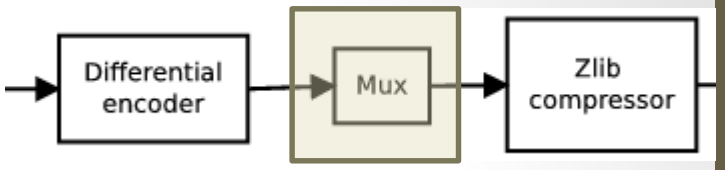
variable	value
?x0	<http://webtlab.it.uc3m.es/_556103110>
?x1	"Wonderboy"
?x2	"2013-02-20T16:58:40Z"
?x3	31317733
?x4	"2013 Women's Cricket World Cup"

  2. Is the triple pattern included in the encoder cache?
     - YES → output differences in bindings wrt. item 1

```
1
<http://webtlab.it.uc3m.es/_556103110>

"2013-02-20T16:58:40Z"
31317733
"2013 Women's Cricket World Cup"
```

  3. Update the LRU cache at the encoder

RDSZ: Algorithm



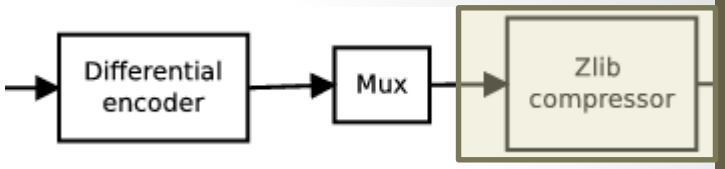
- The multiplexer takes as input…
  - For item 1: its Turtle serialization

```
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix wtl: <http://webtlab.it.uc3m.es/> .

wtl:_556103084 dc:date "2013-02-20T16:58:32Z";
    dc:author "Wonderboy";
    wtl:pageid 6227038;
    wtl:title "Villeroy & Boch" .
```

  - For item 2: the differences wrt. item 1

```
1
<http://webtlab.it.uc3m.es/_556103110>

"2013-02-20T16:58:40Z"
31317733
"2013 Women's Cricket World Cup"
```

- The multiplexer concatenates these item representations
  - Using a delimiter to mark their limits



RDSZ: Algorithm

- The output of the multiplexer is compressed with Zlib
  - Note that the result is text, Zlib is good at taking advantage of redundancies in text
- The binary output from Zlib is sent to the network socket

- The decompressor simply carries out the inverse process…
  - Decompresses Zlib
  - Splits the different items at the demultiplexer
  - Decodes the encoded items
    - The decoder has also an LRU cache keep in sync with that of the encoder

Evaluation

- Datasets
  - Datasets: heterogeneous
    - AEMET1 & AEMET2 (weather)
    - Identica (micro-blogging)
    - Wikipedia (stream of Wikipedia edits)
    - Petrol (credit card transactions)
    - LOD (Linked Observation Data) (weather)
    - Mix (random mixture of items from the former)

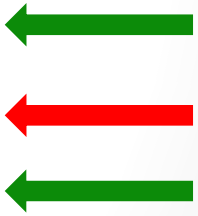
Evaluation

- Setup:
  - Prototype: Python 2.7.3 + RDFLib 4.0.1
  - PC: Ubuntu 12.04, Intel Core 2, 2.53GHz, 8GB RAM
  - Evaluation parameters:
    - cacheSize (size of LRU cache of past items)
    - batchSize (number of items combined at the Mux)
      - Related with the delay perceived at receptor
  - Performance metrics:
    - Compression gain
    - Average processing time per item

Evaluation

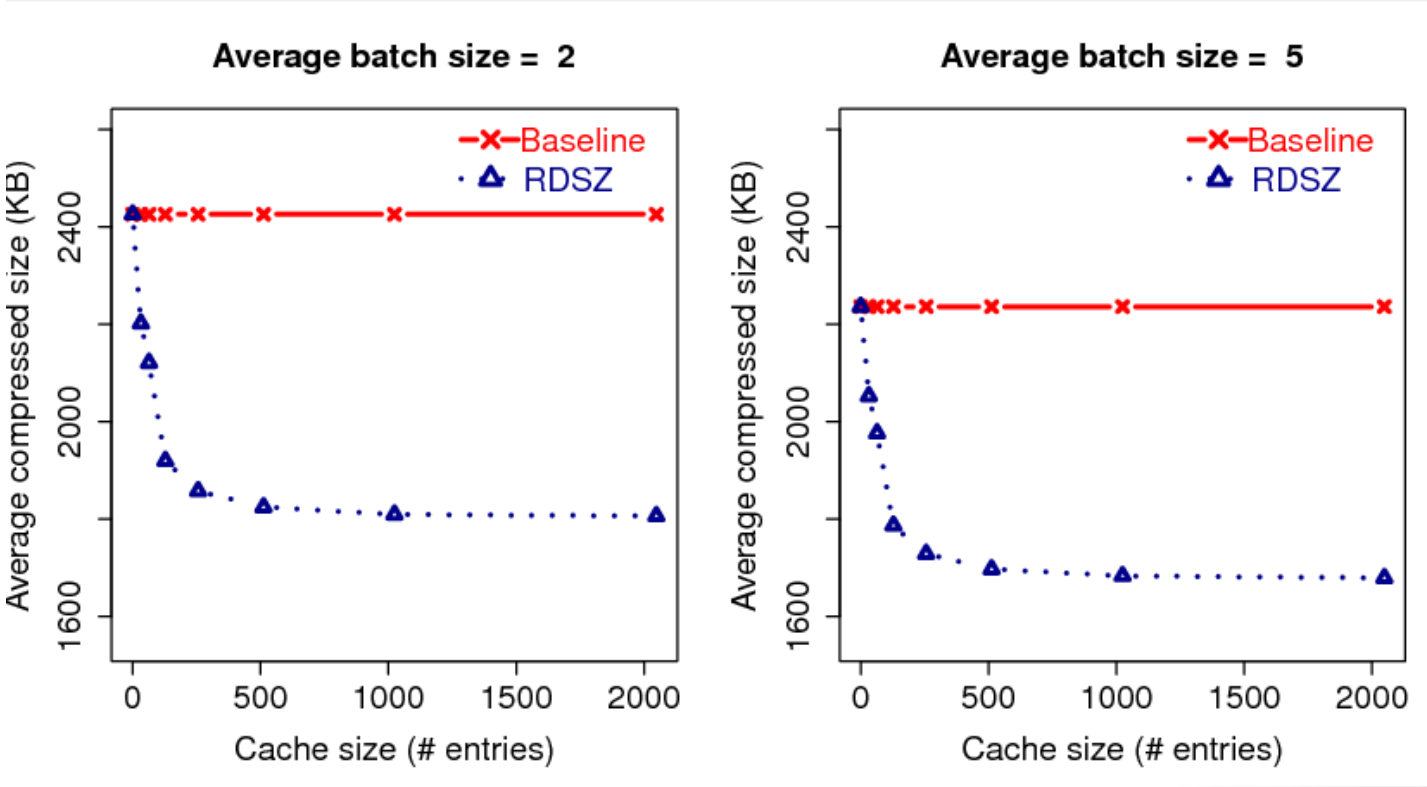
- Compression performance
  - Experiment 1: cacheSize=100, batchSize=5 (fixed)
    - Gain over raw Turtle (no compression): around 90%
    - Gain over Turtle + Zlib: 0% -- 31%

Dataset	Zlib baseline (No differential encoding)				RDSZ	RDSZ
	XML size (bytes)	Turtle size (bytes)	N-Triples size (bytes)	JSON-LD size (bytes)	size (bytes)	Gain
AEMET1	4,325,202	2,299,308	5,552,972	3,051,049	**1,876,624**	18.38%
AEMET2	12,854,321	8,120,614	16,930,673	9,432,218	**5,633,763**	30.62%
Identica	3,335,047	2,604,441	3,569,338	3,078,349	**2,266,868**	12.96%
Wikipedia	3,017,381	**2,466,633**	3,450,287	2,821,515	**2,466,633**	0%
Petrol	29,370,624	23,594,420	34,368,532	25,689,604	**19,806,835**	16.05%
LOD	1,230,708	784,298	1,652,188	1,056,188	**537,646**	31.45%
Mix	889,410	665,786	1,019,060	804,721	**599,775**	9.91%



Evaluation

- Compression performance
  - Experiment 2: variable cacheSize & batchSize in AEMET1



Evaluation

- Response time
  - Total (comp. + decomp.) average processing time per item in the range of milliseconds (8-200ms/item)
    - Worse than the Zlib baseline (ratio 2-3)
  - Linear model suggests linear dependency on the number of triples per item
    - Small items (avg. number of triples 7 -- 179)

Conclusions & Future Lines

- Conclusions
  - RDSZ better compression ratios than Zlib …
    - Taking advantage of structural similarities seems beneficial
  - … but the response time is worse
    - Python prototype not optimized for response time
- Future lines
  - RDSZ needs to be decompressed to query the stream
  - Integrate RDSZ into Ztreamy

Thank you for your attention! (or your tweets)

?