COMP9319 Web Data Compression and Search
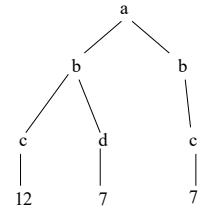
Semistructured / Tree Data,
XML,
XML Compression

1

XPath evaluation
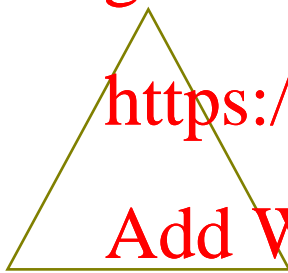
<a><c>12</c><d>7</d><c>7</c>

/ a / b [c = "12"]
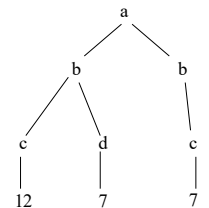
```
        a
      /   \
     b     b
    / \     \
   c   d     c
   |   |     |
  12   7     7
```

2

Query evaluation

Top-down

Bottom-up

Hybrid

3

XPath evaluation

<a><c>12</c><d>7</d><c>7</c>

/ a / b [c = "12"]

```
        a
      /   \
     b     b
    / \     \
   c   d     c
   |   |     |
  12   7     7
```

4

XPath evaluation

<a><c>12</c><d>7</d><c>7</c>

/ a / b [c = "12"]

<c>12</c><d>7</d>

```
        a
      /   \
     b     b
    / \     \
   c   d     c
   |   |     |
  12   7     7
```
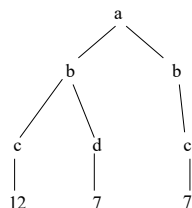
5

Path indexing

- Traversing graph/tree almost = query processing for semistructured / XML data
- Normally, it requires to traverse the data from the root and return all nodes X reachable by a path matching the given regular path expression
- Motivation: allows the system to answer regular path expressions without traversing the whole graph/tree
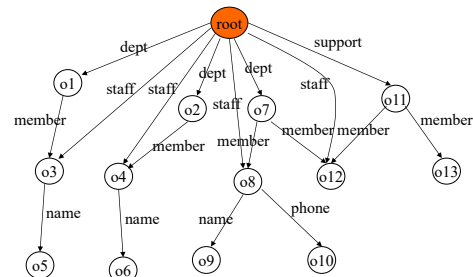
6

Major Criteria for indexing

- Speed up the search (by cutting the search space down)
- Relatively smaller size than the original data graph/tree
- Easy to maintain (during data loading during updates)

An Example of DAG Data

Index graph based on language-equivalence

- a reduced graph that summarizes all paths from the root in the data graph
- The paths from root to o12
  - staff
  - dept/member
  - support/member

Language-equivalent nodes

- Let $L(x) := \{w \mid \exists$ a path from the root to x labeled w$\}$
- The set $L(x)$ may be infinite when there are cycles
- Nodes x, y are language-equivalent ($x \equiv y$) if $L(x) = L(y)$
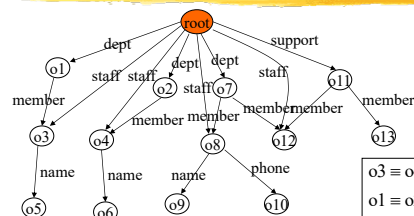- We construct index I by taking the nodes to be the equivalent classes for $\equiv$

Language-equivalent

- The paths from root to o3
  - staff
  - dept/member
- Paths to o4 happen to be exactly the same 2 sequences
- Same for o8 and o12
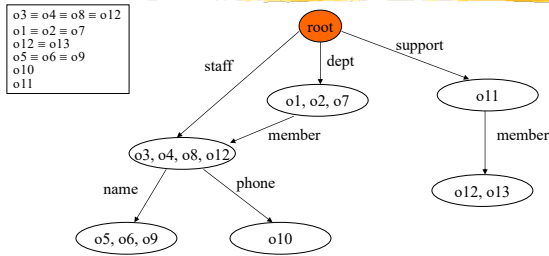- $o3 \equiv o4 \equiv o8 \equiv o12$

Equivalence classes

$o3 \equiv o4 \equiv o8 \equiv o12$	
$o1 \equiv o2 \equiv o7$	
$o12 \equiv o13$	
$o5 \equiv o6 \equiv o9$	
$o10$	
$o11$	

The index graph

o3 ≡ o4 ≡ o8 ≡ o12
o1 ≡ o2 ≡ o7
o12 ≡ o13
o5 ≡ o6 ≡ o9
o10
o11

root — staff — dept — support

dept → o1, o2, o7
support → o11
member → o12, o13
staff → o3, o4, o8, o12
member → o3, o4, o8, o12
name → o5, o6, o9
phone → o10

13

Query processing based on the index graph

root — staff — dept — support

dept → o1, o2, o7
member → o3, o4, o8, o12
name → o5, o6, o9
phone → o10
support → o11
member → o12, o13

dept/member/(name | phone)

-> dept/member/name UNION dept/member/phone
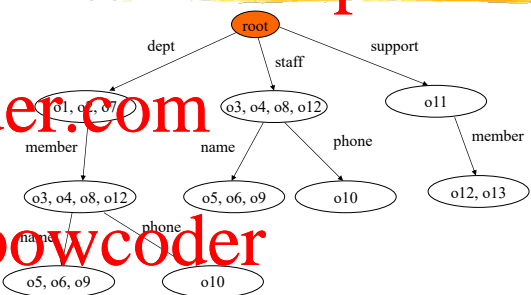
-> {o5, o6, o9} UNION {o10}

-> {o5, o6, o9, o10}
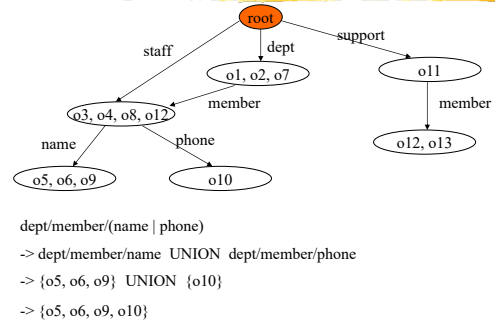
14

About this indexing scheme

- The index graph is never > the data
- In practice, the index graph is small enough to fit in memory
- Construct the index is however a problem
  - check two nodes are language-equivalent is very expensive (are PSPACE)
  - approximation based on bisimulation exists

15

A Data Guide

root — dept — staff — support

dept → o1, o2, o7
member → o3, o4, o8, o12
name → o5, o6, o9
phone → o10
staff → o3, o4, o8, o12
name → o5, o6, o9
phone → o10
support → o11
member → o12, o13

16

About Data Guide

- unique labels at each node
- (hence) extents are no longer disjoint
- query processing proceeds as before
- size of the index may >= data size
- good for data that is regular & has no cycles

17

XML-Specific Compressors

- Unqueriable Compression (e.g. XMill):
  - Full-chunked: data commonalities eliminated
  - Very good compression ratio
- Queriable Compression (e.g. XGrind, XPRESS):
  - Fine-grained: data commonalities ignored
  - Inadequate compression ratio and time
  - Support simple path queries with atomic predicate

18

XMill

- **First specialized** compressor for XML data
  - <u>SAX parser</u> (a fast, event-based parser) for parsing XML data
  - Still using gzip as its underlying compressor
  - Clever grouping of data into containers for compression

- Compress XML via **three** basic techniques
  - Compress the structure separately from the data
  - Group the data values according to their types
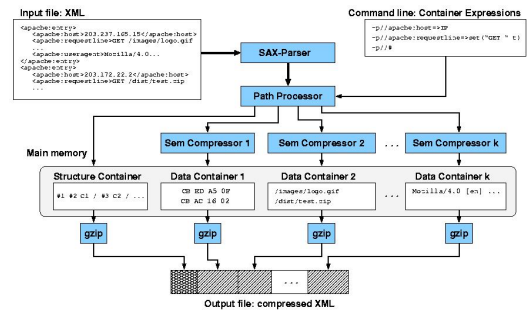 - Apply semantic (specialized) compressors:

19

XMill Architecture:

Figure 4: Architecture of the Compressor

20

An Example:Web Server Logs

ASCII File 15.9 MB (gzipped 1.6MB):

202.239.238.16|GET / HTTP/1.0|text/html|200|1997/10/01-00:00:02|-|4478|-|-|http://www.net.jp/|Mozilla/3.1[ja](I)

XML-ized <u>apache web log</u> inflates to 24.2 MB (gzipped 2.1MB):

```
<apache:entry>
  <apache:host> 202.239.238.16 </apache:host>
  <apache:requestLine> GET / HTTP/1.0 </apache:requestLine>
  <apache:contentType> text/html </apache:contentType>
  <apache:statusCode> 200</apache:statusCode>
  <apache:date> 1997/10/01-00:00:02</apache:date>
  <apache:byteCount> 4478</apache:byteCount>
  <apache:referer> http://www.net.jp/ </apache:referer>
  <apache:userAgent> Mozilla/3.1$[$ja$]$(I)</apache:userAgent>
</apache:entry>
```

21

How Xmill Works: Three Ideas

Compress the structure separately from the data:

gzip Structure

```
<apache:entry>
  <apache:host> </apache:host>
  ...
</apache:entry>
```

+

gzip Data

```
202.239.238.16
GET / HTTP/1.0
text/html
200
...
```

=1.75MB

22

How Xmill Works: Three Ideas

Group the data values according to their types:

gzip Structure

```
<apache:entry>
...
</apache:entry>
```

+

gzip Data1

```
202.23.23.16
224.42.24.55
...
```

+

gzip Data2

```
GET / HTTP/1.0
GET / HTTP/1.1
...
```

=1.33MB

23

How Xmill Works: Three Ideas

Apply semantic (specialized) compressors:

gzip Structure + gzip c1(Data1) + gzip c2(Data2) + ...   =0.82MB

Examples:
- 8, 16, 32-bit integer encoding (signed/unsigned)
- differential compressing (e.g. 1999, 1995, 2001, 2000, 1995, ...)
- compress lists, records (e.g. 104.32.23.1 → 4 bytes)

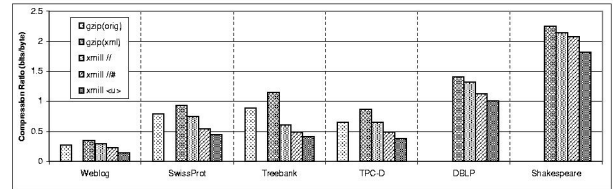Need user input to select the semantic compressor

24

Experiments

Data Source	Original Size	Size in XML	Regular?	Depth	Tags	DataGuide Size
Weblog Data	57.7MB	172MB	yes	1	10	11
SwissProt	98.5MB	158MB	yes	3	92	58
Treebank	39.6MB	53.8MB	no	35	251	339920
TPC-D	34.6MB	119MB	yes	2	43	60
DBLP	-	47.2MB	yes	3	10	145
Shakespeare	-	7.3MB	no	5	21	58

25

XML Compression

26

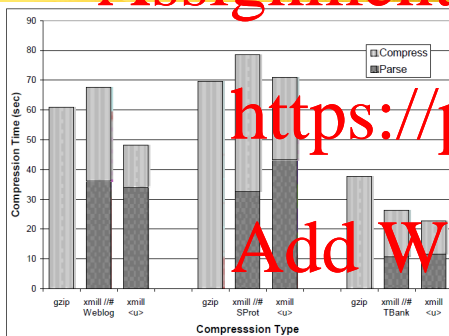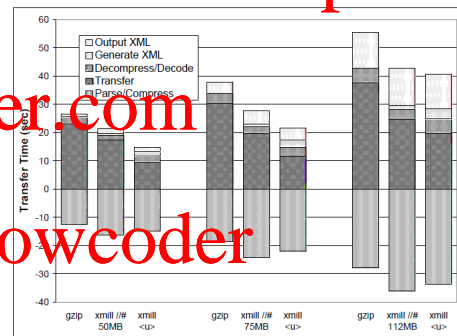## Compression Time

27

Transfer Time (& Decode)

28

XGRIND (Tolani & Haritsa, 2002)

- Encodes elements and attributes using XMill's approach
- **DTD-conscious:** enumerated attributes with k possible values are encoded using a $\log_2 k$-bit scheme
- Data values are encoded using non-adaptive Huffman coding
  - Requires two passes over the input document
  - Separate statistical model for each element/attribute
- Homomorphic compression: compressed document retains original structure

29

XGRIND

Original Fragment:

```
<student name="Alice">
    <a1>78</a1>
    <a2>86</a2>

    <midterm>91</midterm>

    <project>87</project>
</student>
```

Compressed Fragment:

```
T0 A0 nahuff(Alice)
    T1 nahuff(78) /
    T2 nahuff(86) /
    T3 nahuff(91) /
    T4 nahuff(87) /
    /
```

30

XGRIND

- Many queries can be carried out entirely in compressed domain
  - Exact-match, prefix-match
- Some others require only decompression of relevant values
  - Range, substring
- Queryability comes at the expense of achievable compression ratio: typically within 65-75% that of XMill

31

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder