Optimizing Pattern Queries for Web Access Logs

Tadeusz Morzy, Marek Wojciechowski, Maciej Zakrzewicz
Poznan University of Technology
Poland

Motivation

- Web access logs represent the history of users' visits to a web server
- Web access logs are becoming the subject of behavior analysis
- Web access logs tend to be large in size (100s megabytes, gigabytes)
- The behavior analysis requires intensive searching of web access logs in order to find users who exhibit a specific access pattern (pattern queries)

Presentation Outline

- Problem formulation: optimizing pattern queries over web access logs
- Sequential index structure: construction, usage and properties
- Experimental results
- Final conclusions

Problem: Web Access Log Example

```
154.11.231.17 - - [13/Jul/2000:20:42:25 +0200] "GET / HTTP/1.1" 200 1673
154.11.231.17 - - [13/Jul/2000:20:42:25 +0200] "GET /apache_pb.gif HTTP/1.1" 200 2326
192.168.1.25 - - [13/Jul/2000:20:42:25 +0200] "GET /demo.html HTTP/1.1" 200 520
192.168.1.25 - - [13/Jul/2000:20:42:25 +0200] "GET /books.html HTTP/1.1" 200 3402
160.81.77.20 - - [13/Jul/2000:20:42:25 +0200] "GET / HTTP/1.1" 200 1673
154.11.231.17 - - [13/Jul/2000:20:42:25 +0200] "GET /car.html HTTP/1.1" 200 2580
192.168.1.25 - - [13/Jul/2000:20:42:25 +0200] "GET /cdisk.html HTTP/1.1" 200 3856
10.111.62.101 - - [13/Jul/2000:20:42:25 +0200] "GET /new/demo.html HTTP/1.1" 200 971
```


192.168.1.25: /demo.html -> /books.html -> cdisk.html

Problem: Data Model and Pattern Query

IP	TS	URL
1	1	A
1	2	B
1	3	C
1	4	D
2	1	A
2	2	E
2	3	C
2	4	F
3	1	B
3	2	C
3	3	D
3	4	A

Pattern Query:
Find all event sequences that contain the given subsequence.

SQL Implementation:

```
SELECT IP
FROM R R1, R R2, R R3
WHERE R1.IP = R2.IP
  AND R2.IP = R3.IP
  AND R1.TS < R2.TS
  AND R2.TS < R3.TS
  AND R1.URL = 'A'
  AND R2.URL = 'E'
  AND R3.URL = 'F'
```

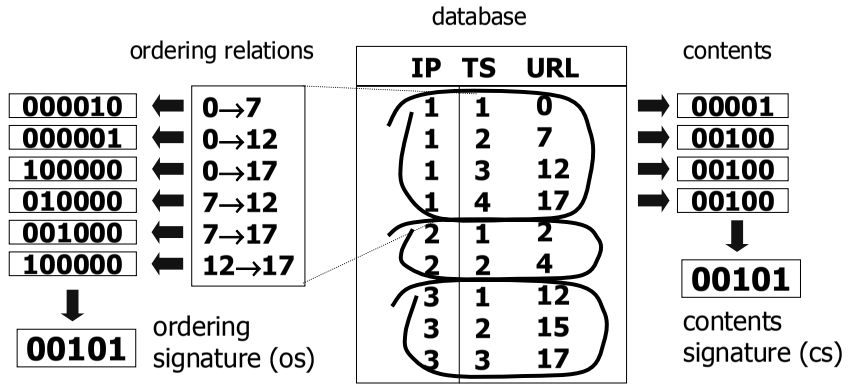
Problem: Efficiency

- Since web access logs tend to be very large, there is a problem of appropriate optimizing the database access while performing pattern queries
- The available indexing techniques (B+ trees, bitmapped, k-d trees, R trees) do not significantly improve pattern queries since:
  - Pattern queries deal with partial matches of multi-tuple sequences
  - Traditional indexing techniques ignore sequential relations between tuples

Sequential Index Structure

- A sequential index is a data structure which helps optimize sequential pattern queries
- A sequential index physically records item ordering inside database sequences
- A sequential index uses bitmaps to represent all its information

Sequential Index Construction and Usage

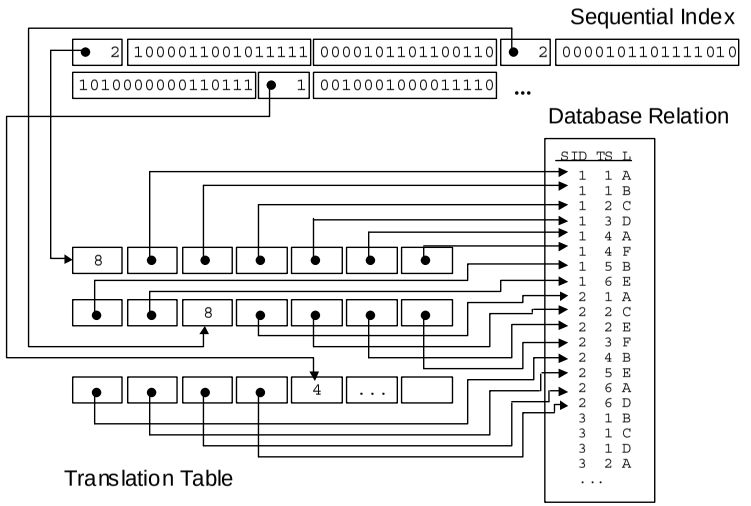


- Signatures property:
  "If (cs(*X*) && cs(*Y*)) = cs(*X*) and (os(*X*) && os(*Y*)) = os(*X*), then the sequence Y contains the sequence X"
- The above property can be used to efficiently perform pattern queries

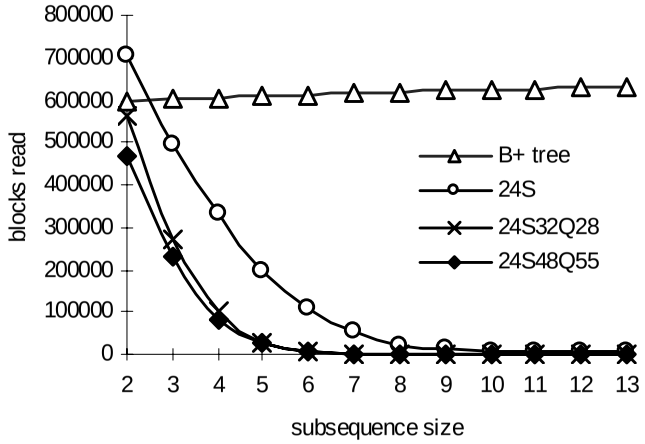
Sequential Index Extensions

- For long sequences, the number of ordering relations is large, therefore long bitmaps are required
- Partitioning the long sequences into smaller subsequences helps keep the bitmap size smaller
  - a separate bitmap is generated for each subsequence
- Partitioning helps maintain the index structure during incremental insert operations

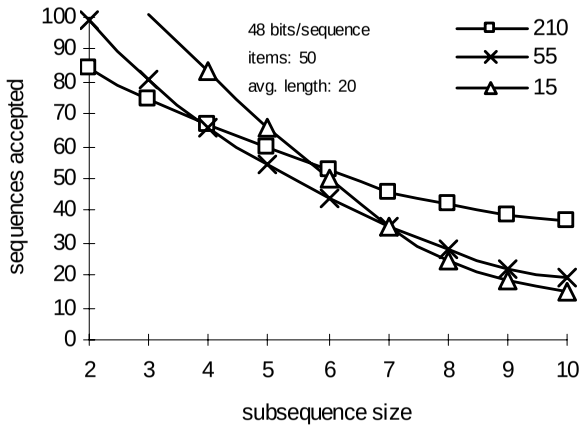
Physical Storage



Experimental Results (1)



Experimental Results (2)



Summary

- Pattern queries over web access logs are commonly used during user behavior analysis
- Pattern queries can be efficiently optimized with sequential indexes
- Sequential indexes store bitmaps that represent both sequence contents and sequence ordering
- Future work: to extend the technique to support regular expression pattern queries