An Injection with Tree Awareness Adding Staircase Join to PostgreSQL

Sabine Mayer° Torsten Grust° Maurice van Keulen• Jens Teubner°

Universität Konstanz

°University of Konstanz, Dept. of Computer and Information Science, Germany
•University of Twente, Faculty of EEMCS, The Netherlands
http://www.inf.uni-konstanz.de/dbis/

University of Twente
The Netherlands

Abstract

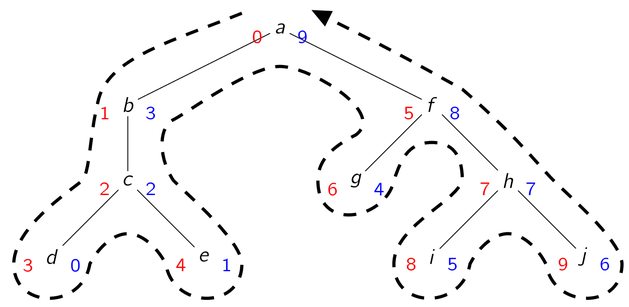
The **XPath accelerator** ("*pre*/*post* numbering") has proven to be an efficient encoding to losslessly store XML data in relational databases. Conventional RDBMSs, however, remain ignorant of interesting properties of the encoded tree data, and make thus no or poor use of these properties. At VLDB 2003 we devised a new join algorithm, **staircase join**, that encapsulates tree-specific knowledge and can turn SQL systems into highly efficient XPath processors.

This demonstration delivers the promise we have made at VLDB 2003: a notion of tree-awareness can be injected into a conventional disk-based RDBMS kernel in terms of staircase join. We feature a side-by-side comparison of both, an original and a staircase join-enhanced instance of **PostgreSQL**. The required changes to PostgreSQL were local. The achieved effect, however, is significant: the demonstration proves that this injection of tree awareness makes PostgreSQL a high-performance XPath processor that closely adheres to the XPath semantics.

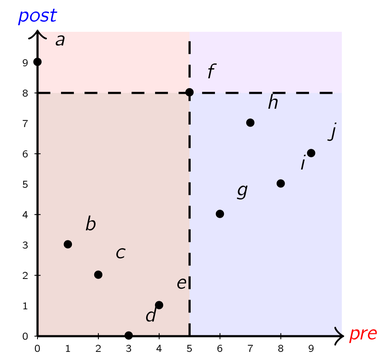
Accelerating XPath Location Steps

```
<a>
  <b>
    <c>
      </d></e>
    </c>
  </b>
  <f>
    </g>
    <h>
      </i></j>
    </h>
  </f>
</a>
```

An XML document.



Assignment of pre/post values.



The pre/post plane. Partitions the document and illustrates range queries.

v	*pre*	*post*
a	0	9
b	1	3
c	2	2
d	3	0
e	4	1
f	5	8
g	6	4
h	7	7
i	8	5
j	9	6

The document table.

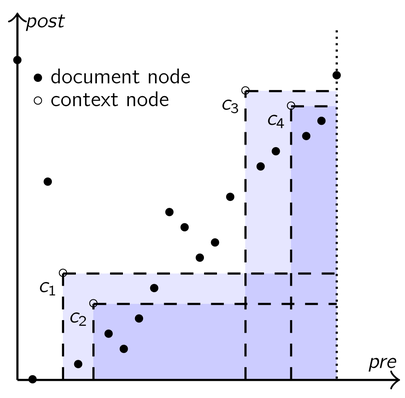
Range Queries:

$n \in c/\texttt{preceding} \Leftrightarrow c.pre > n.pre \wedge c.post > n.post$
$n \in c/\texttt{following} \Leftrightarrow c.pre < n.pre \wedge c.post < n.post$
$n \in c/\texttt{descendant} \Leftrightarrow c.pre < n.pre \wedge c.post > n.post$
$n \in c/\texttt{ancestor} \Leftrightarrow c.pre > n.pre \wedge c.post < n.post$

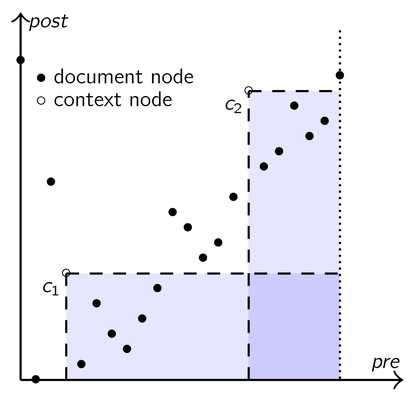
$context/\texttt{following/descendant} \Leftrightarrow$

```
SELECT DISTINCT n2.*
  FROM context c, document n1, document n2
 WHERE c.pre < n1.pre AND c.post < n1.post
   AND n1.pre < n2.pre AND n1.post > n2.post
 ORDER BY n2.pre;
```

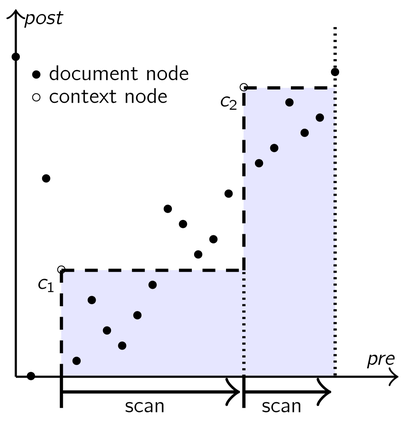
Staircase Join: Pruning, Partitioning, and Skipping



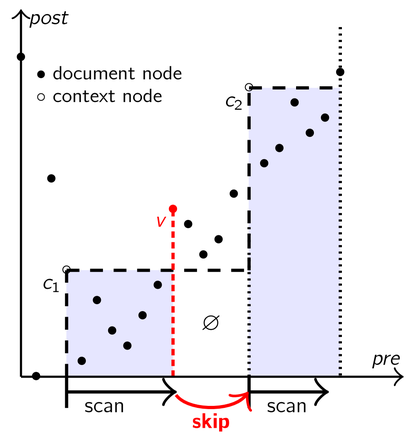
A `descendant` location step in the pre/post plane.



After pruning.



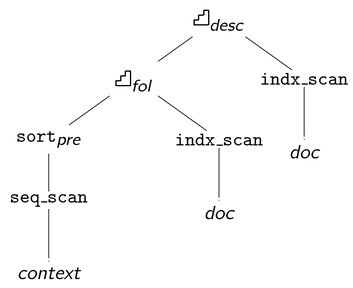
Partitioning.



Skipping.

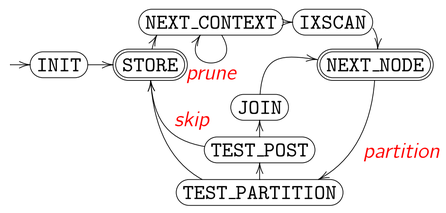
Staircase Join in a Relational Database

- Behaves like any join.
- Allows for rewriting.
- Chosen during dynamic programming.



Execution Plan of an XPath Query.

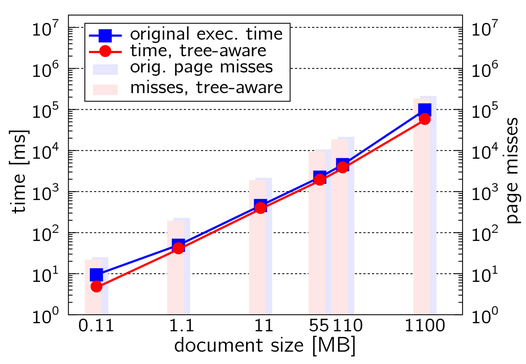
- Works as a state automaton.
- Relies on existing index structures (B-tree).
- Allows for streaming/pipelining.



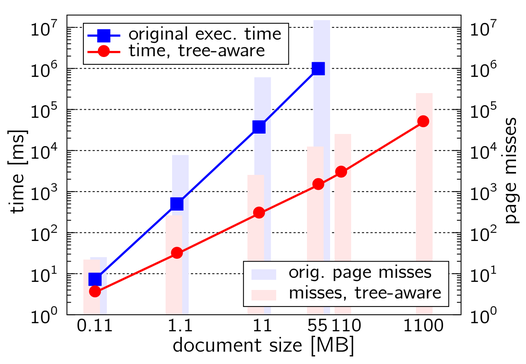
The `descendant` axis state automaton.

Performance Tests in PostgreSQL

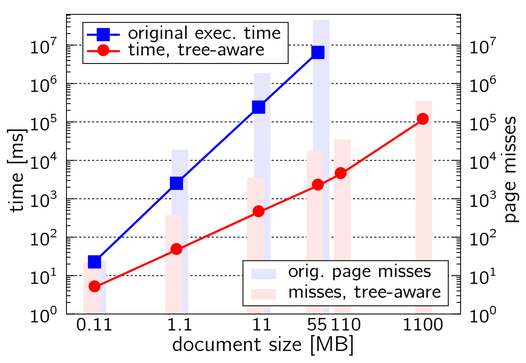
Execution times and index-related buffer statistics in the tree-aware and the original PostgreSQL DBMS:



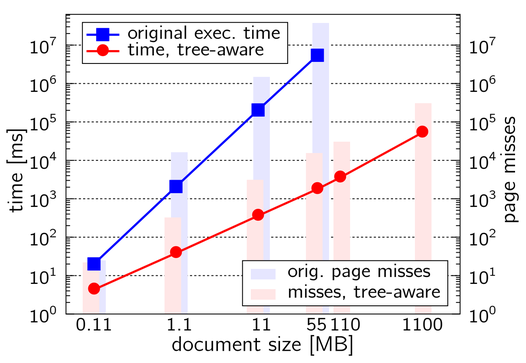
Query: $/\texttt{descendant::}t_1/\texttt{descendant::}t_2$



Query: $/\texttt{descendant::}t_1/\texttt{ancestor::}t_2$



Query: $/\texttt{descendant::}t_1/\texttt{preceding::}t_2$



Query: $/\texttt{descendant::}t_1/\texttt{following::}t_2$