XML Storage and Indexing

Web Data Management and Distribution

Serge Abiteboul Ioana Manolescu Philippe Rigaux
Marie-Christine Rousset Pierre Senellart

Web Data Management and Distribution
http://webdam.inria.fr/textbook

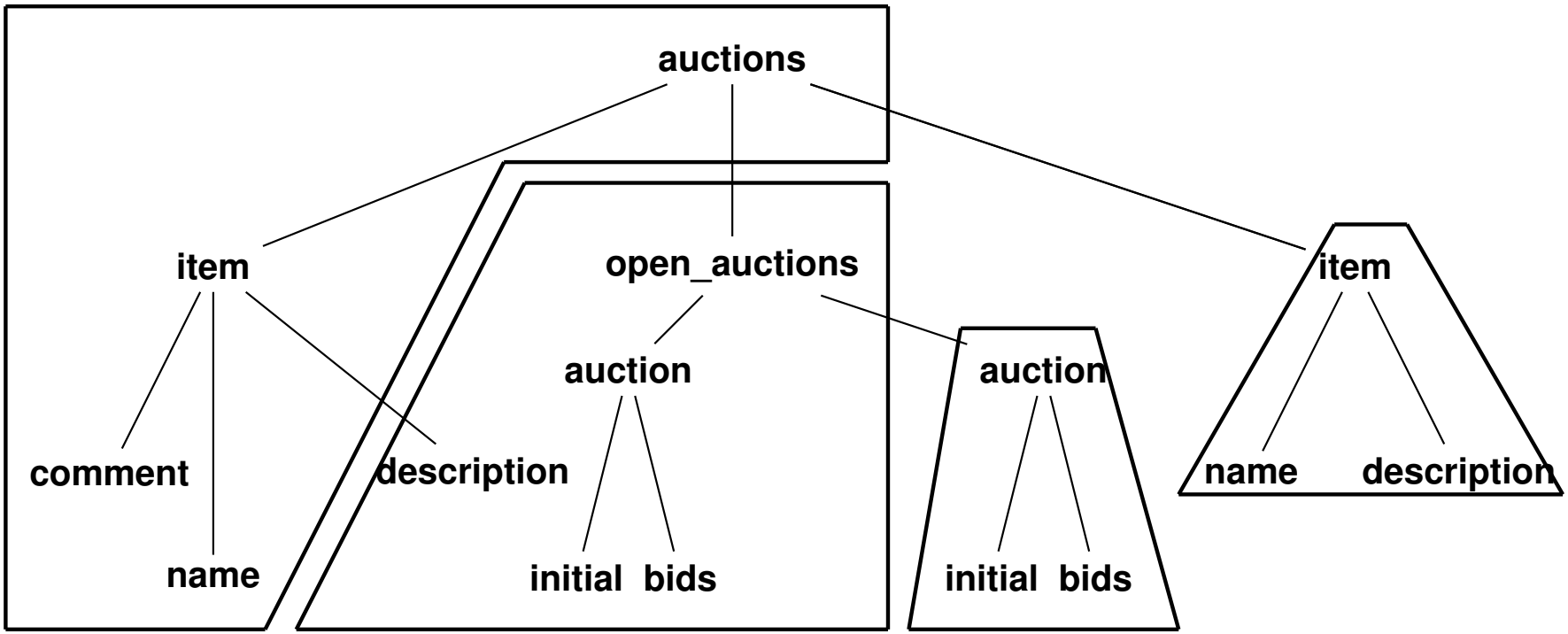
September 29, 2011

Outline

Motivation

- PTIME algorithms for evaluating XPath queries:
  - Simple tree navigation
  - Translation into logic
  - Tree-automata techniques
- These techniques assume XML data in memory
- Very large XML documents: what is critical is the number of disk accesses

Naïve page-based storage



Processing Simple Paths

- `/auctions/item` requires the traversal of two disk pages;
- `/auctions/item/description` and `/auctions/open_auctions/auction/initial` both require traversing three disk pages;
- `//initial` requires traversing all the pages of the document.

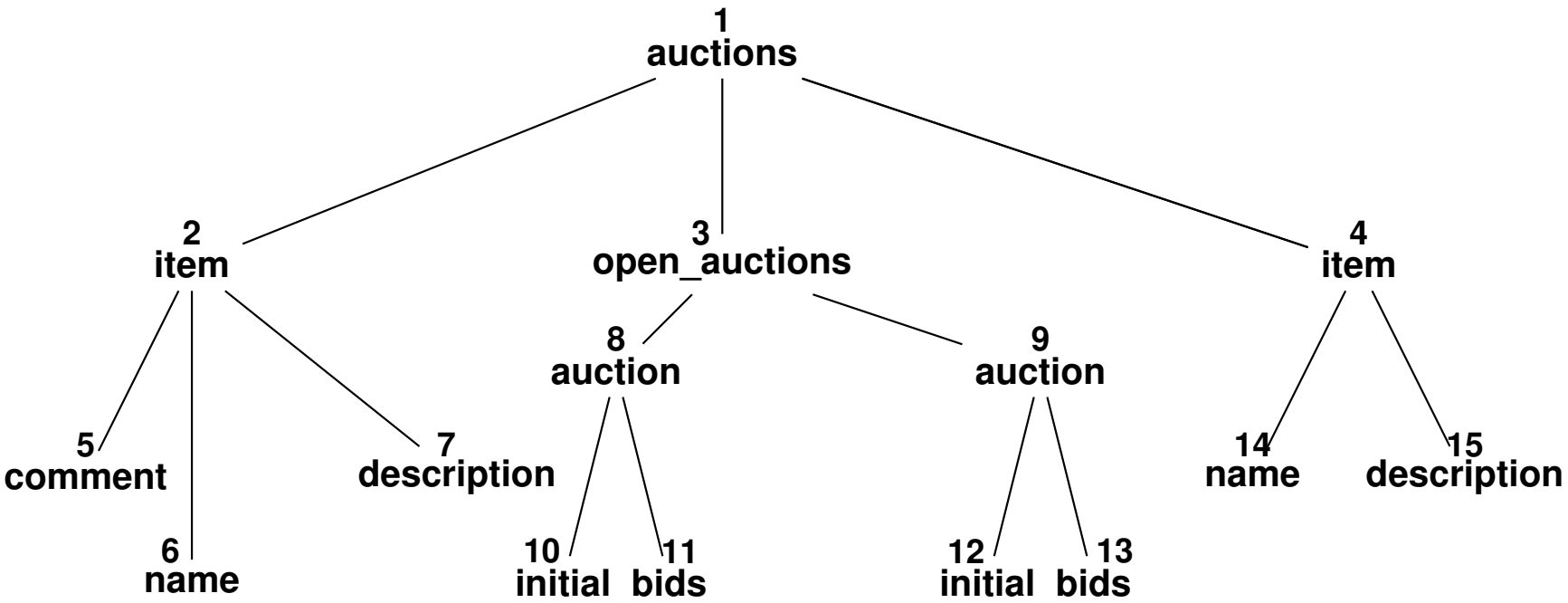
Remedies

Smart fragmentation. Group nodes that are often accessed simultaneously, so that the number of pages that need to be accessed is globally reduced.

Rich node identifiers. Sophisticated node identifiers reducing the cost of join operations.

Outline

Simple node identifiers



Partial instance of the *Edge* relation

pid	cid	clabel
-	1	auctions
1	2	item
2	5	comment
2	6	name
2	7	description
1	3	open_auctions
3	8	auction
...	...	...

Processing XPath queries with simple IDs

`//initial` Direct index lookup.

$$\pi_{cid}\left(\sigma_{clabel=initial}\left(Edge\right)\right)$$

`/auctions/item` A join is needed.

$$\pi_{\text{cid}}\left(\left(\sigma_{clabel=auctions}\left(Edge\right)\right) \bowtie_{cid=pid} \left(\sigma_{clabel=item}\left(Edge\right)\right)\right)$$

`//auction//bid` Unbounded number of joins!

`//auction/bid`	$\pi_{cid}\left(A \bowtie_{cid=pid} B\right)$
`//auction/*/bid`	$\pi_{cid}\left(A \bowtie_{cid=pid} Edge \bowtie_{cid=pid} B\right)$
`//auction/*/*/bid`	$\cdots$

Outline

Tag-partitioned *Edge* relations

auctionsEdge

pid	cid
-	1

itemEdge

pid	cid
1	2
1	4

open_auctionsEdge

pid	cid
1	3

auctionEdge

pid	cid
3	8
3	9

Reduces the disk I/O needed to retrieve the identifiers of elements having a given tag. Partitioning of queries with // steps in non-leading position remains as difficult as before.

Outline

Path-partitioned storage

/auctions

pid	cid
-	1

/auctions/item

pid	cid
1	2
1	4

/auctions/item/name

pid	cid
2	6
4	14

Paths

path
/auctions
/auctions/item
/auctions/item/comment
/auctions/item/name
...

Query evaluation using path partitioning

`//item//bid` can be evaluated in two steps:

- Scan the `path` relation and identify all the parent-child paths matching the given linear XPath query;
- For each of the paths thus obtained, scan the corresponding path-partitioned table.

Many branches still require joins across the relations.
For very structured data, the `path` relation is typically much smaller than the data set itself. Thus, the cost of the first processing step is likely negligible, while the performance benefits of avoiding numerous joins are quite important.

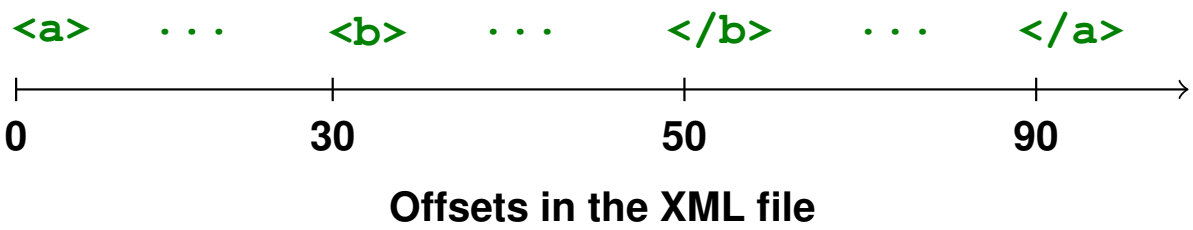
Outline

Identifiers

Within a persistent XML store, each node must be assigned a unique identifier (or ID, in short).
We want to encapsulate structure information into these identifiers to facilitate query evaluation.

Outline

Region-based identifiers



The so-called region-based identifier scheme simply assigns to each XML node n, the pair composed of the offset of its begin tag, and the offset of its end tag. We denote this pair by $(n.begin, n.end)$.

- the region-based identifier of the <a> element is the pair (0, 90);
- the region-based identifier of the <b> element is pair (30, 50).

Using region-based identifiers

Comparing the region-based identifiers of two nodes n_1 and n_2 allows deciding whether n_1 is an ancestor of n_2. Observe that this is the case if and only if:

- $n_1.start < n_2.start$, and
- $n_2.end < n_1.end$.

No need to use byte offset:

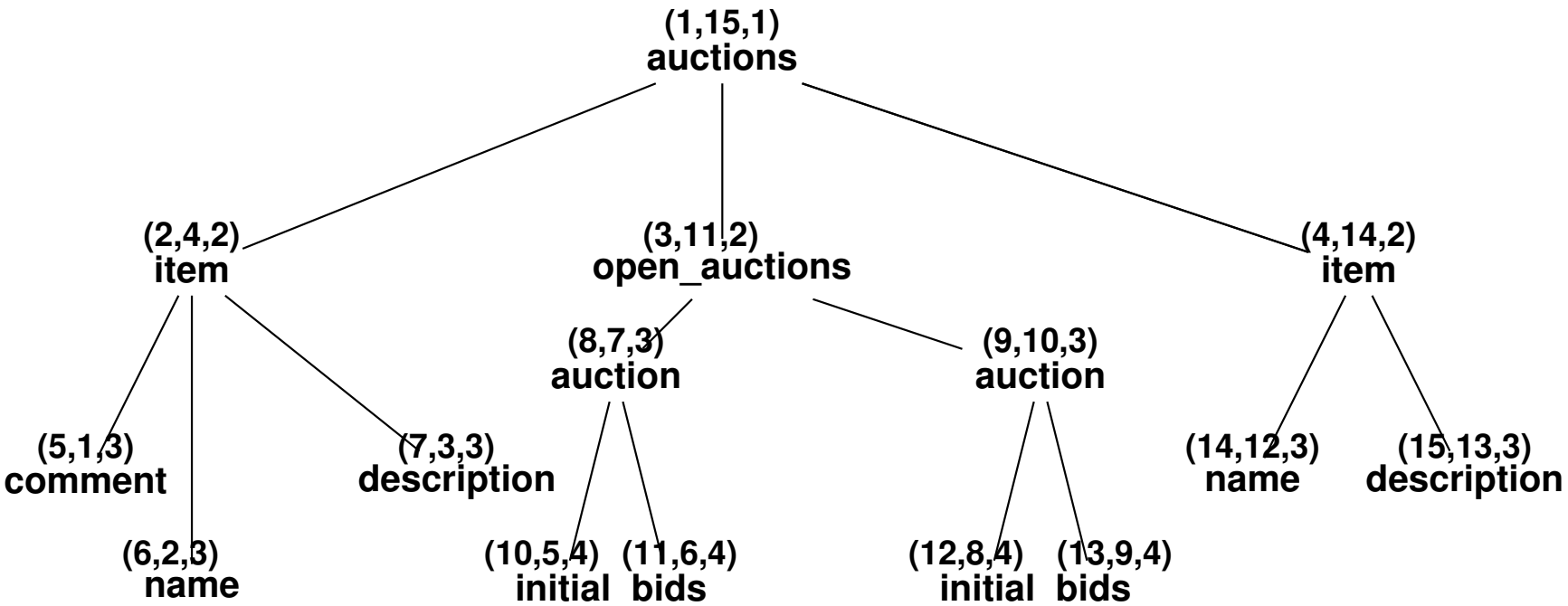
(Begin tag, end tag). Count only opening and closing tags (as one unit each) and assign the resulting counter values to each element.

(Pre, post). Preorder and postorder index (see next slide).

(Pre, post, depth). The depth can be used to decide whether a node is a parent of another one.

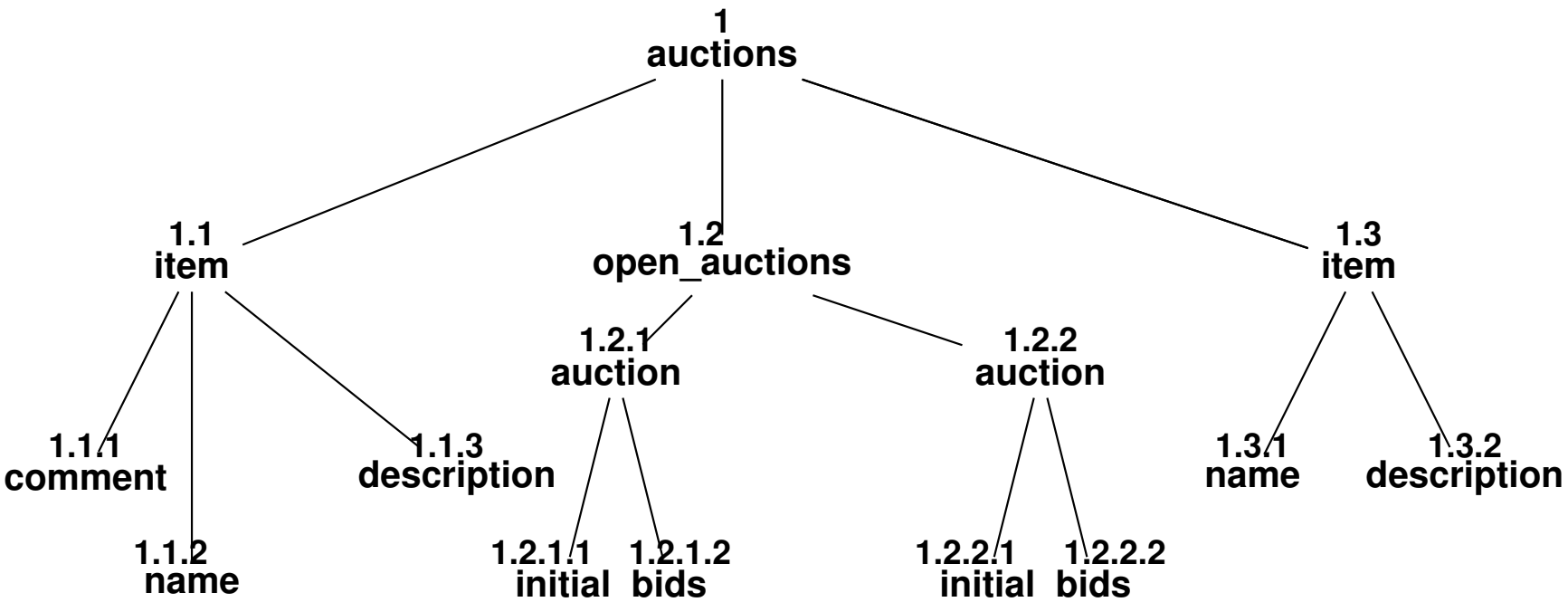
Region-based identifiers are quite compact, as their size only grows logarithmically with the number of nodes in a document.

(pre, post, depth) node identifiers



Outline

Dewey node identifiers



Using Dewey identifiers

- Ancestor decided by looking if an ID is a prefix of another one (largest non-identical prefix for parent).
- Possible to decide preceding-sibling, following-sibling, preceding, following.
- Given two Dewey IDs n_1 and n_2, one can find the ID of the *lowest common ancestor (LCA)* of the corresponding nodes. The ID of the LCA is the longest common prefix of n_1 and n_2. Useful for keyword search.

Main drawback: length (large, variable) of the identifiers.

Outline

Re-labeling

Consider a node with Dewey ID 1.2.2.3. Suppose we insert a new first child to node 1.2. Then the ID of node 1.2.2.3 becomes 1.2.3.3.
In general:

- Offset-based identifiers need to be updated as soon as a character is inserted or removed in the document.
- (start, end), (pre, post), Dewey IDs need to be updated when the elements of the documents change.
- Possible to avoid relabeling on deletions, but gaps will appear in the labeling scheme.
- Relabeling operation quite costly.