XML as a Boxwood Data Structure

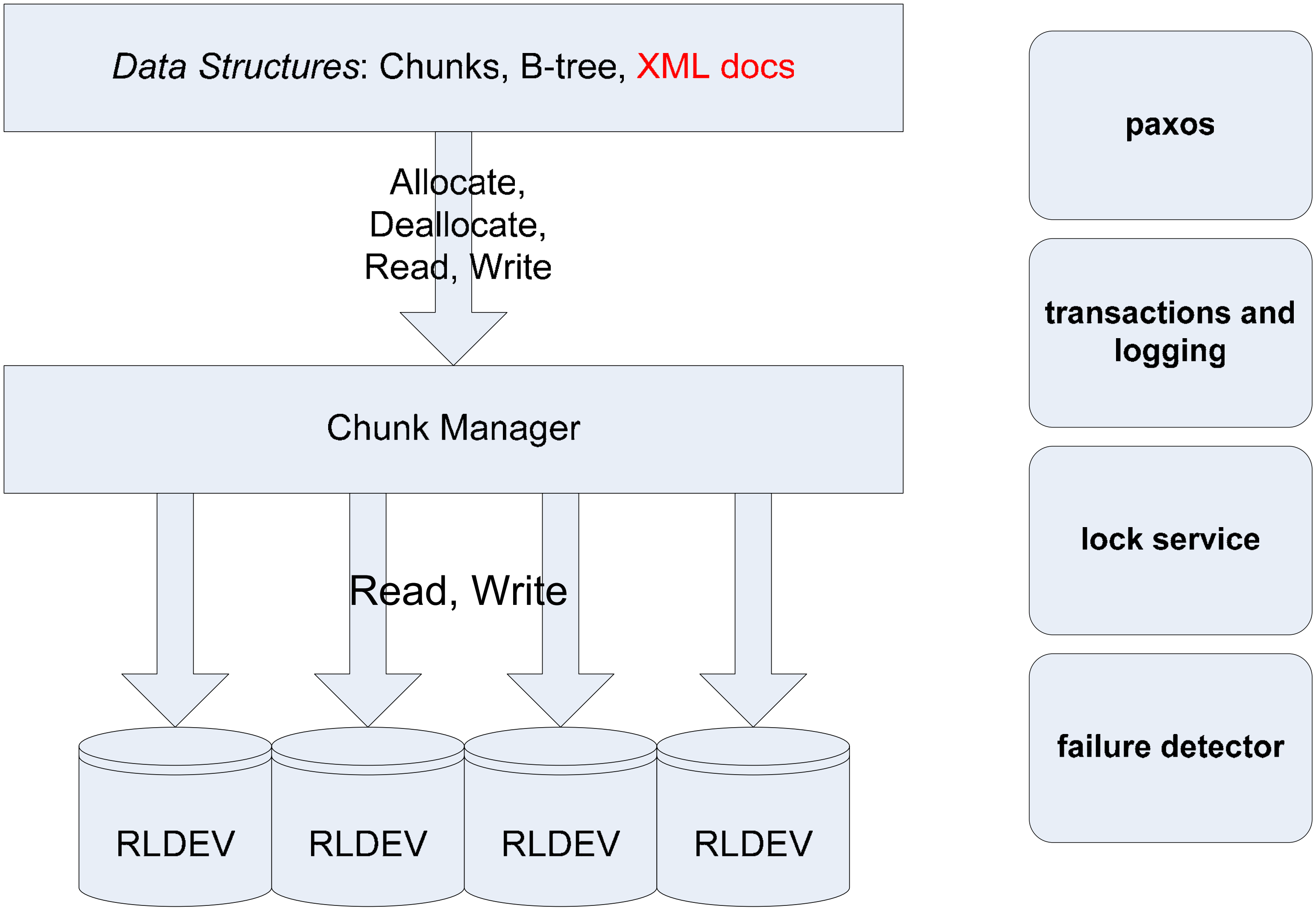
Feng Zhou, John MacCormick, Lidong Zhou,
Nick Murphy, Chandu Thekkath

8/20/04

Overview

- Goal: Make XML a native data structure of the Boxwood storage virtualization system
- Motivation:
  - Useful for users of Boxwood because XML is gaining popularity as a way of storing and exchanging data
  - Validates the argument that the modular Boxwood design facilitates construction of complex data structures
  - Validates the design and implementation of the Boxwood prototype

Boxwood overview



XML example

```
<?xml version="1.0" encoding="UTF-8"?>
<collection>
  <recipe>
    <title>Beef Parmesan with Garlic Angel Hair
  Pasta</title>
    <ingredient name="beef cube steak" amount="1.5"
  unit="pound"/>
    <preparation>
      <step>
       Preheat oven to 350 degrees F (175 degrees C).
      </step>
    </preparation>
    <nutrition calories="1167" fat="23"
  carbohydrates="45" protein="32"/>
  </recipe>
</collection>
```

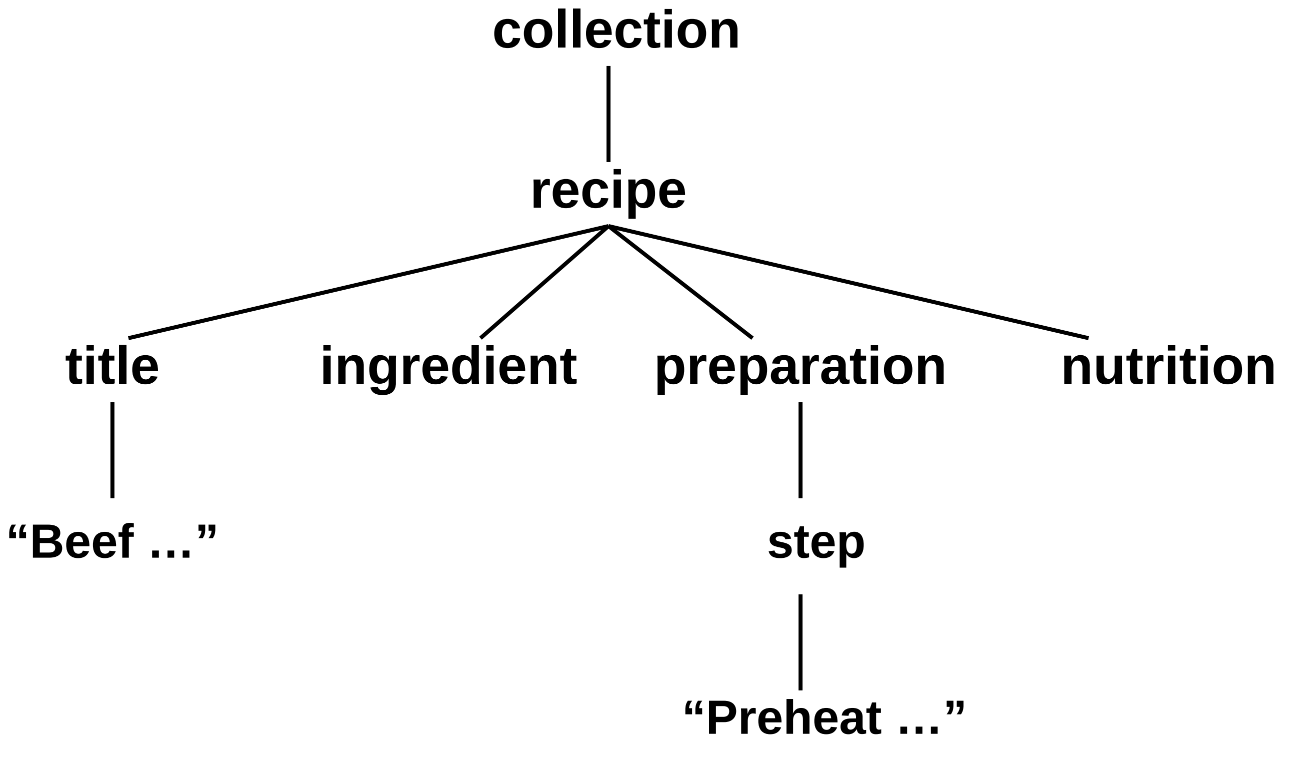
elements

attributes

text fragment

All called nodes in the XML data model

XML as a tree


collection
recipe
title
ingredient
preparation
nutrition
“Beef …”
step
“Preheat …”

Challenges in building a scalable XML store

- Must support flexible access methods
  - XPath: file system path like queries
    /collection/recipe[nutrition/@calories<1000]/title
  - XQuery: more complex SQL like queries
- Large amount of documents
- Large documents
- Document mutation (e.g. node insertions/removals)

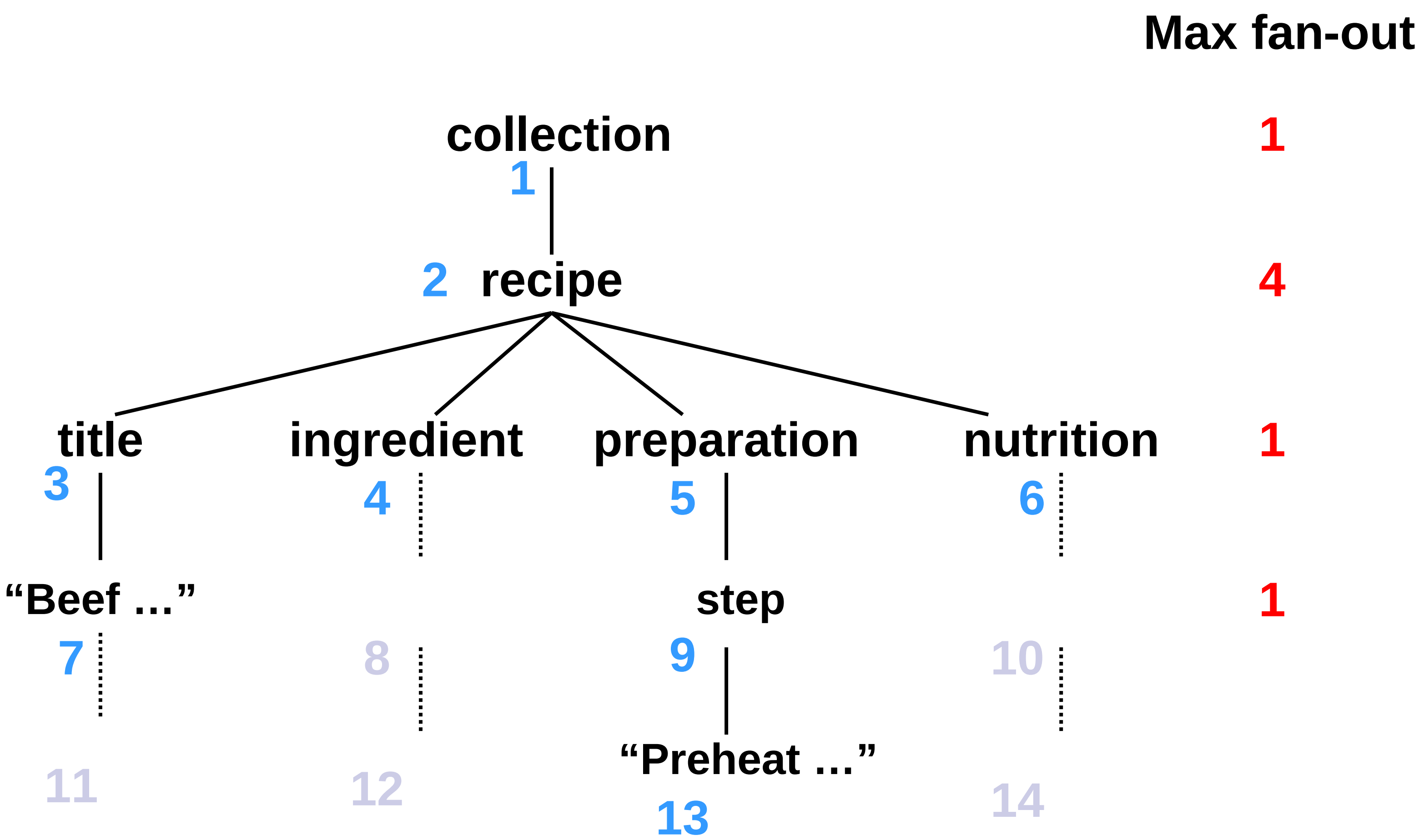
Outline

- Overview
- XML to Boxwood mapping
- XPath query processing
- Evaluation

Numbering Scheme

- XPath requires fast access to related nodes: parent/children/sibling
- Could be done by maintaining pointers to these nodes but too much space overhead
  - At least 4 pointers needed (parent, first child, next sibling, previous sibling)
- XML db community proposed schemes to number nodes in regular ways that make these pointers unnecessary.
- We use the numbering scheme used in the eXist XML database.

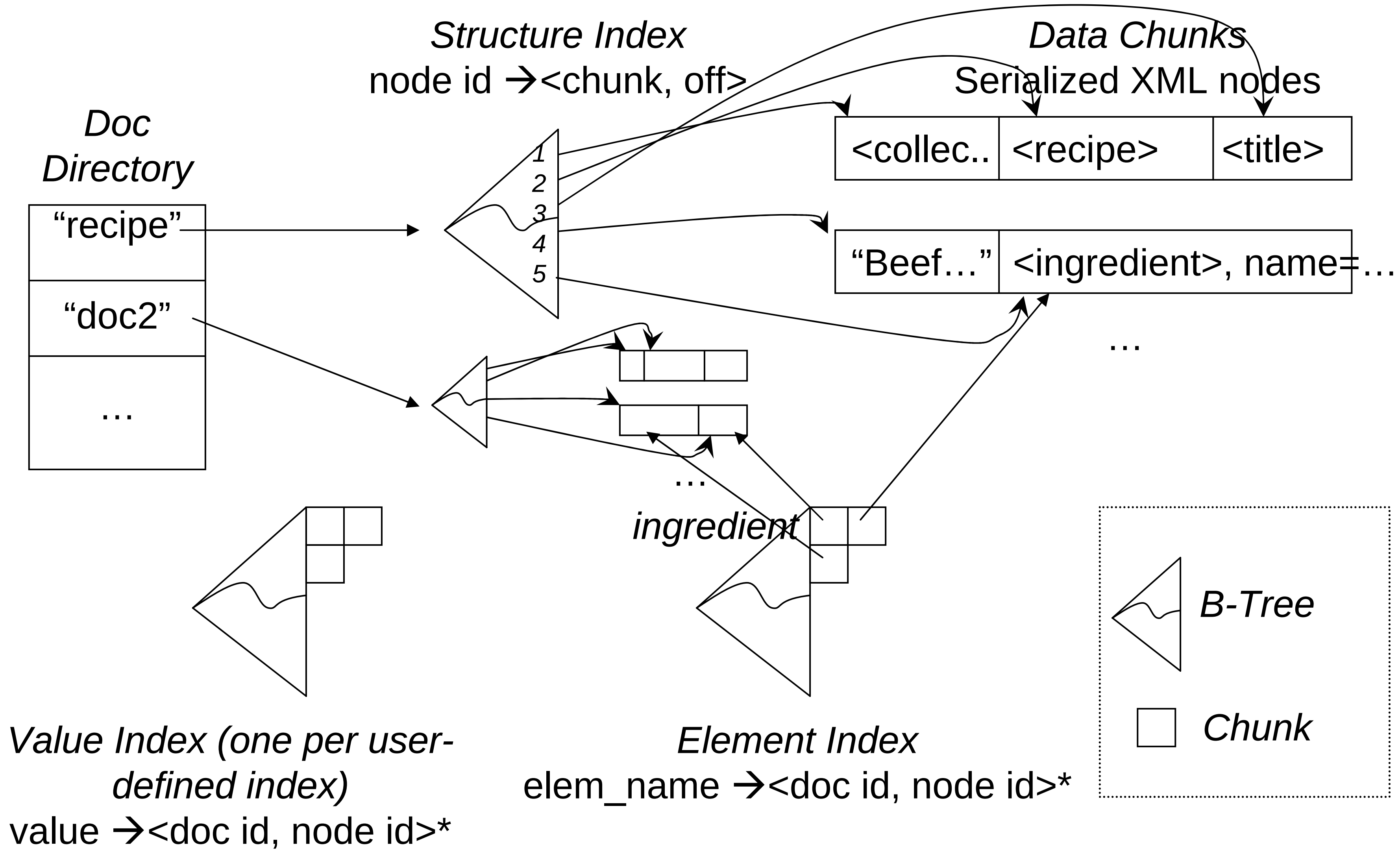
Numbering nodes in the XML tree



Mapping to Boxwood

- Basic Boxwood data structures
  - Chunks (variable-sized, “persistent malloc()”)
  - B-Tree (fixed-sized keys → fixed-sized values)

XML store data structures



Outline

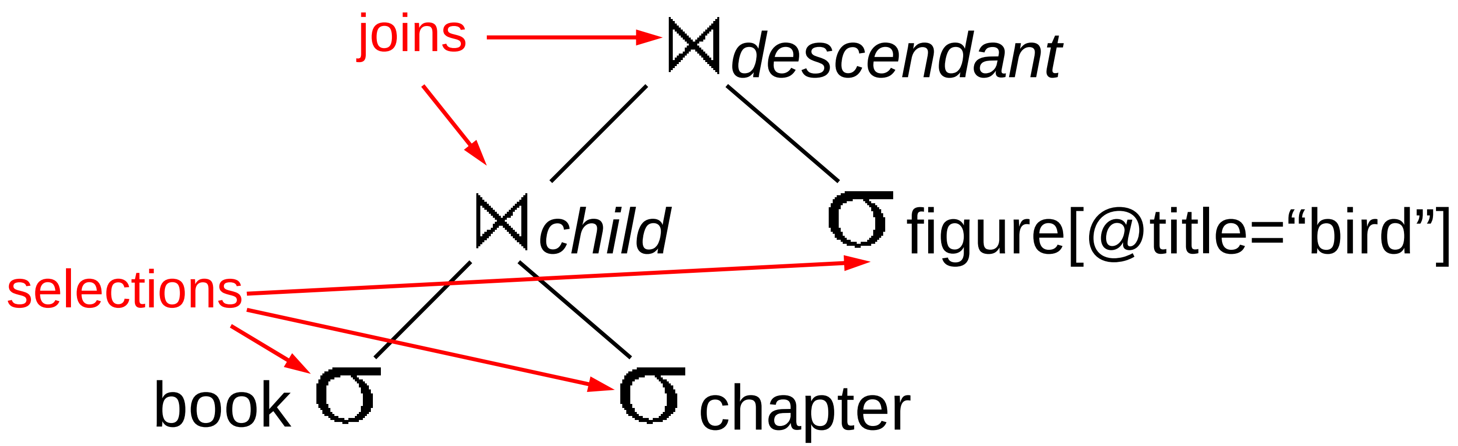
- Overview
- XML to Boxwood mapping
- XPath query processing
- Performance and experience

XPath query processing

- /collection/recipe[nutrition/@calories<1000]/title
- XPath expressions
  - ***Path expressions:*** nutrition/@calories
  - Comparison expressions: nutrition/@calories<1000
  - Arithmetic expressions
  - …

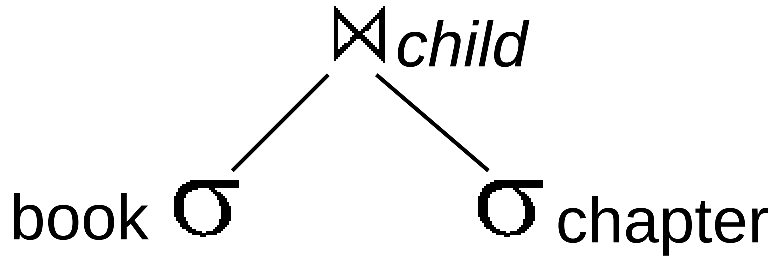
Evaluating path expressions

- Uses recently proposed XML query processing techniques (Li&Moon, VLDB'01)
- //book/chapter//figure[@title="bird"] =



- No query optimization yet

Join algorithms



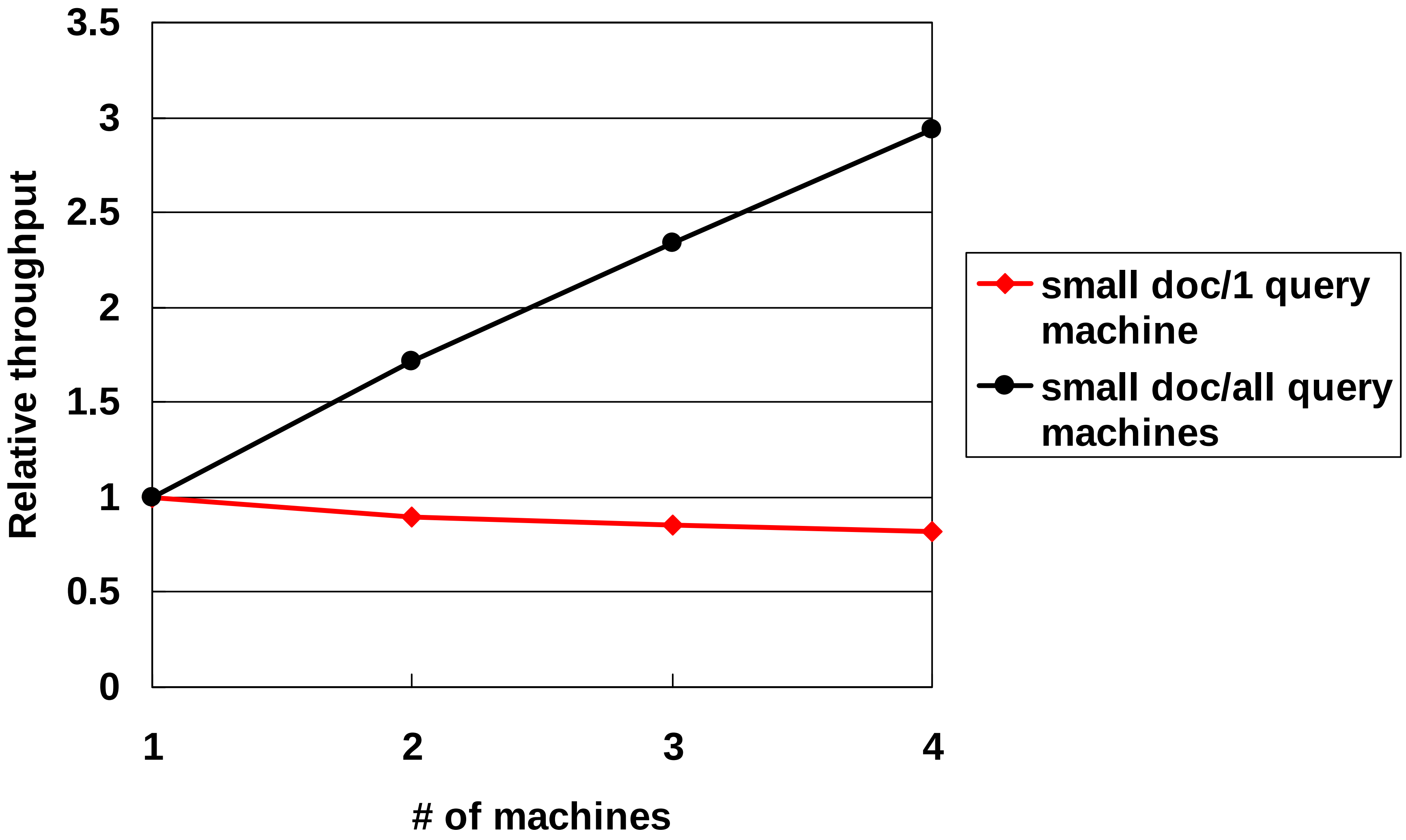
- //book/chapter
- **Nested loop**:
  1. Find “book” elements by enumerating all elements
  2. For each “book” element, enumerate all children and output the ones named “chapter”
- **“Path join”**: efficient join operations utilizing indexes
  1. A={IDs of all elements named “book”}, obtained through the element index
  2. B={IDs of all elements named “chapter”}
     child axis join:
  3. Result= $\{x \mid x \in B, \mathrm{parent}(x) \in A\}$

- “child” join is one of the 13 different join operations

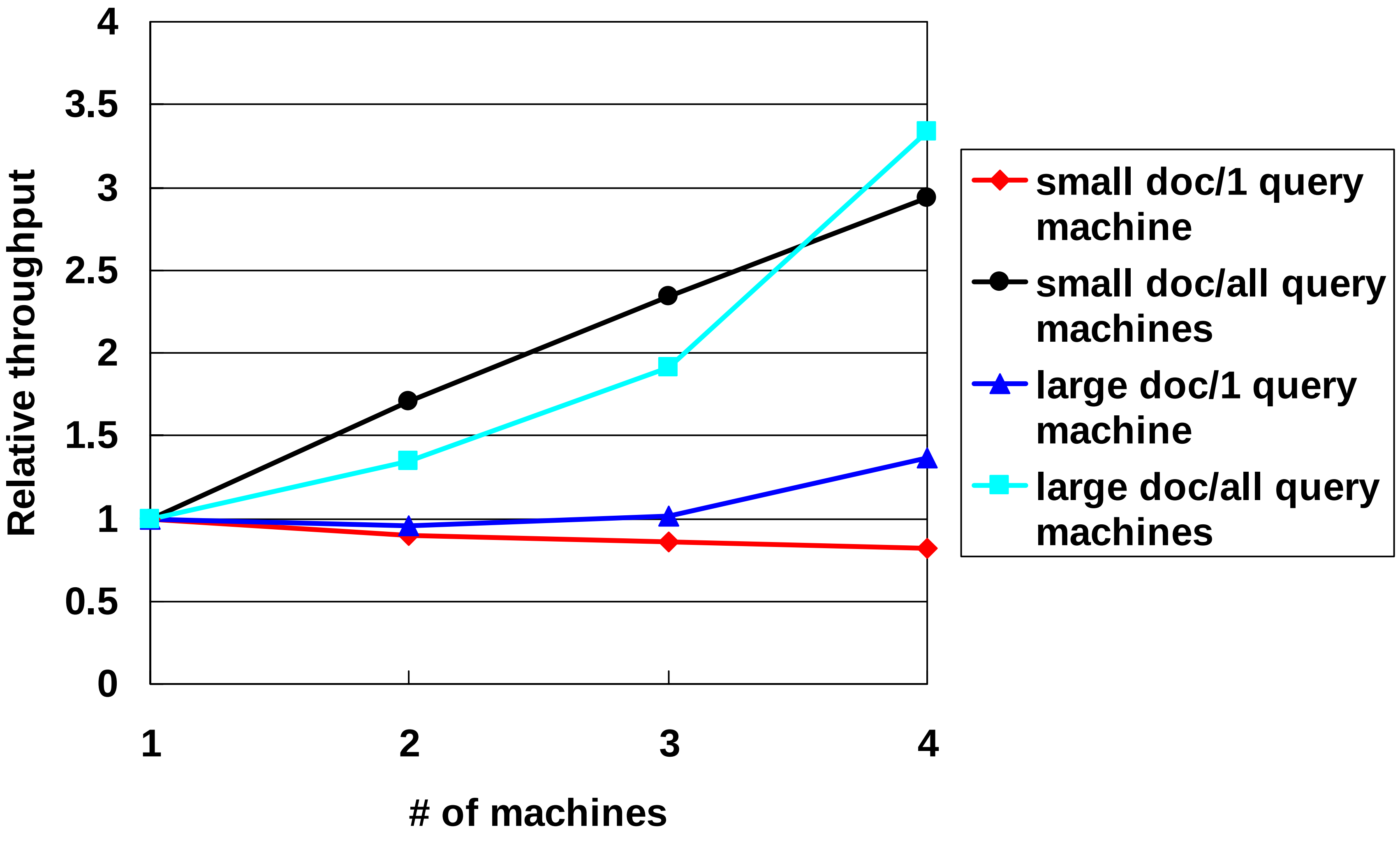
Performance Results

- Single machine
  - Inserting an XML doc containing 47K nodes (SIGMOD conference records) into the store: 75 sec
  - Read the whole doc: 0.2 sec
  - A simple XPath (selecting all papers of a certain author): 1.2 sec (whole doc in cache)
- Scalability experiment
  - # of machines: 1-4 machines
  - Local cache size: 10K nodes
  - Document size: 10K / 40K XML nodes
  - Issue XPath queries from one machine, or from all machines in parallel

Scalability Experiment Results



Scalability Experiment Results



Issues using Boxwood

- Bulk-loading a B-tree is slow
  - Could support a “bulk-loading” mode where coarser-grained locking & recovery is used
- B-tree only supports fix-sized keys and values
  - The index implements a dictionary with variable-size keys and values on top of B-tree
  - Could be generalized
- Various performance issues
  - e.g. a flag set when opening the device file causing all i/o to be serialized

Boxwood benefits

- Chunk and B-tree abstracts data layout and distribution
- Data consistency easier
  - B-Trees already does proper locking
  - Others manually call global lock server
- Recovery support easier
  - B-Tree operations already transactional
  - Provide logging and recovery code in other cases
- Free data caching in B-tree and chunks
- 7K LOC, about 3K for XPath, reasonably easy to write

In this project, we

- devised a way of mapping XML to existing Boxwood data structures
- implemented data structures and algorithms to efficiently access and query XML documents
- demonstrated that Boxwood facilitates building of scalable data structures.
- made significant improvements to several aspects of Boxwood performance

Thank you!

Status

- Non-validating (no DTD or XML Schema support) XML parser
- XmlReader (.NET framework) interface for sequential access to docs
- Custom interface for random access
- Subset of XPath 2.0
  - Supports: all path expressions axes, predicates, all arithmetic and comparison expressions
  - Missing: variables, functions, control flow

Data vs. Document XML

- Data XML
  - e.g. sales orders, stock quotes, scientific data
  - more structured, fine-grained (smallest unit of data could be a number), order generally not significant
- Document XML
  - e.g. books, emails, Word documents in XML
  - less regular, coarse-grained (smallest unit could be a paragraph), lots of mixed content