A Fast Index for Semistructured Data

Brian F. Cooper, Neal Sample, Michael J. Franklin,
Gisli R. Hjaltason, Moshe Shadmon

Zhihui Wang
CPS 296.1
March 28, 200

Overview

- Motivation
- Index Fabric
- Indexing XML with the Index Fabric
- Experiments
- Conclusion

Motivation

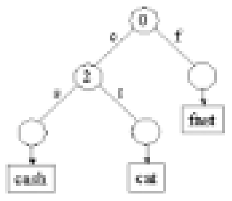
- To leverage existing relational database technology and provide much better performance than pervious approaches to query semi-structured data

A Fast Index for Semistructured Data

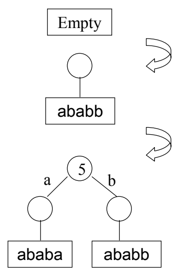
- Motivation
- **Index Fabric**
  - Patricia Tries based
  - Layered approach
  - Two kinds of links
  - Insertion, Deletion, and Update
- Indexing XML with the Index Fabric
- Experiments
- Conclusion

Index Fabric --- Patricia Tries based

- The nodes are labeled with their depth: the character position in the key represented by the node
  - The size of the Patricia trie does not depend on the length of inserted keys
  - Key compression is lossy



Patricia Tries



Insert a key “ababb”

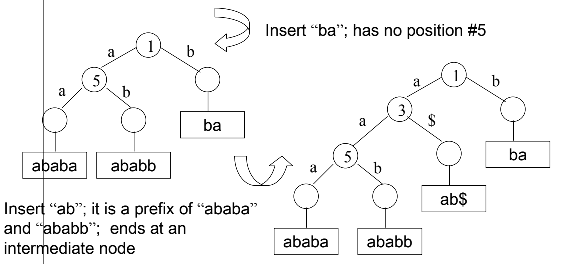
Insert “ababa”; search ends at babb~=ababa

1st difference is at position #5

Insert “ba”; has no position #5

“http://www.csse.monash.edu.au/~lloyd/tildeAlgDS/Tree/PATRICIA.html”

Patricia Tries



Insert "ab"; it is a prefix of "ababa" and "ababb"; ends at an intermediate node

Append a special terminating character, for example, `$‘, which is only allowed to appear at the ends of keys

"http://www.csse.monash.edu.au/~lloyd/tildeAlgDS/Tree/PATRICIA.html"

Index Fabric --- Layered approach

- Index Fabric improves Patricia tries and make it balanced and optimized for disk-based access like B-tree
- Each query accesses the same number of layers
- The index can have as many layers as necessary, the highest layer always contains one block
- The keys are stored very compactly, and blocks have a very high out-degree. In practice, three layers is enough to store billions of keys

Index Fabric --- Two kinds of links

- Labeled far links
  - Like normal edges in a trie, but connects a node in one layer to a subtrie in the lower layer
- Unlabeled direct links
  - Connects a node in one layer to a block with a node representing the same prefix in the lower layer

Index Fabric --- Searching

1. Begin in the root node
2. Within a block, comparing characters in search key to edge labels, and following the corresponding edge
3. If a labeled edge is far link, the search proceeds to a subtrie in the lower layer
4. If no labeled edge matches the appropriate character of the search key, the search follows a direct edge into new block in the lower layer

Index Fabric --- Searching

5. The search will reach the layer 0 finally, either the desired data are found, or no match indicates the key does not exist
6. Verify the found data if matches the search key, because of the lossy compression of the Patricia trie

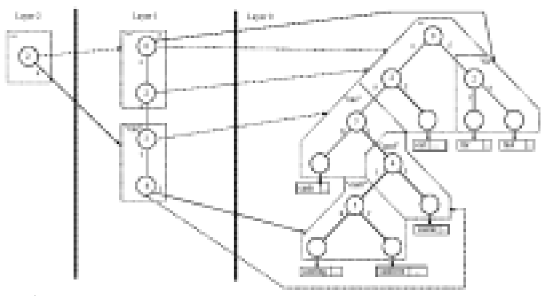
Note:

When searching, it is possible to enter the wrong block, then have to backtrack.

Index Fabric --- Insertion, Deletion, and Update

- Inserting a key involves a change to a block in the lowest layer. If the block is full, it will be split. In some rare cases, split can propagate till the highest layer, and a new horizontal layer will be generated in that case.
- For a deletion, first locate the key in a block, then remove the edge pointing to the leaf for the deleted key. If the block is underutilized, block recombination may be required.
- A update can be processed as a key deletion followed by a key insertion.

Index Fabric --- An example



A Fast Index for Semistructured Data

- Motivation
- Index Fabric
- **Indexing XML with the Index Fabric**
  - Designators
  - Raw Paths
  - Refined Paths
  - Storage Manager
  - Accelerating Queries
- Experiments
- Conclusion

Indexing XML with the Index Fabric --- Designators

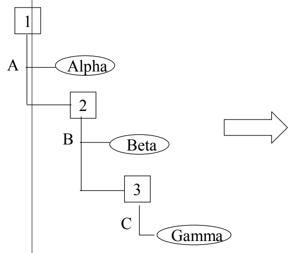
- Designator
  - A unique special character or characters assign to each tag in XML
- Designator dictionary
  - Maintain the mapping between designators and XML tags
- Insert the designator-encoded XML string into Index Fabric
- XML Tags in queries are translated into designators, and to form a search key over the Index Fabric

Indexing XML with the Index Fabric --- Raw Paths

- Index the hierarchical structure of the XML by encoding a root-to-leaf path as a string
- Treat attribute like tagged children, but use different designators to distinguish the same name tag and attribute
- Can use alternate designators to encode the ordering of tags in the XML documents

Raw Paths --- An simple example

<A>Alpha<B>Beta<C>Gamma</C></B></A>



X alpha

X Y Beta

X Y Z Gamma

Indexing XML with the Index Fabric --- Refined Paths

- Refine paths are specialized paths through XML that optimize frequent access patterns
- Support queries that have wildcards, alternates, and different constants
- DBA decides which refined paths are appropriate
- Both raw paths and refined paths are stored in the same index and accessed using string lookup

Refined Paths --- An Example

Find the XML segments in which <B1> tag and <B2> tag are siblings:

```
Select x
From *.B1 x, x.B2
```

Assign a designator “Z” to such a path

```
<A>
    <B1> Beta </B1>
    <B2> Sigma </B2>
 </A>
```

=> Key: Z Beta Sigma

Indexing XML with the Index Fabric --- Refined Paths

- Two steps to create a refined path index:
  - The XML documents are parsed to extract information matching the access pattern of the refined path;
  - The information is encoded as an Index Fabric key and inserted into the index.

Indexing XML with the Index Fabric --- storage manager

- Not dictate a particular architecture for the storage manager of the database system
- In order to leverage the maturity of relational database system, both the index blocks and the XML data are stored in a relational database

Indexing XML with the Index Fabric --- Accelerating Queries

- Simple path expression
  - Specifies a sequence of tags starting from the root of the XML
  - *Key lookup operator* to search for the raw path key
- General path expression
  - Allow for alternates, optional tags, and wildcards
  - Expand the query into multiple simple path expressions

Indexing XML with the Index Fabric --- Accelerating Queries

- Examples of general path expressions:
  - A.(B1|B2).C => A.B1.C and A.B2.C
  - A.(%)*.C
    - *Prefix key lookup* search for “A” prefix
    - Follow each child of “A” to see if there is a “C” somewhere down below
- Refined paths can further optimized queries

A Fast Index for Semistructured Data

- Motivation
- Index Fabric
- Indexing XML with the Index Fabric
- **Experiments**
  - Setup
  - Results
- Conclusion

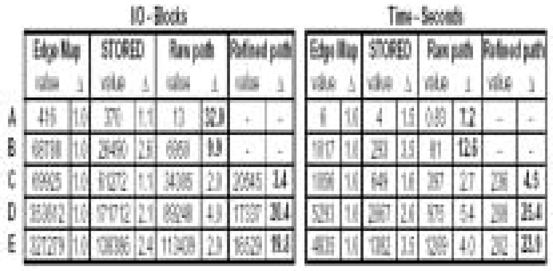
Experiments --- "Apples to apples"

- Store the XML data set in a popular commercial relational database system
- Compare the performance of queries using the DBMS' native B-tree index versus using the Index Fabric implemented on top of the same database system

Experiment --- Setup

- Data set: DBLP
- Two methods to store XML data, and indexed by the RDBMS' B-tree native indexing mechanism
  - Edge-mapping
  - STORED mapping
- Five queries

Experiment --- Results



IO - Blocks

	Edge Map		STORED		Raw path		Refined path	
	value	i	value	i	value	i	value	i
A	416	1.0	370	1.1	13	32.0	-	-
B	68788	1.0	26490	2.6	6950	9.9	-	-
C	69925	1.0	61272	1.1	34305	2.0	20545	3.4
D	353862	1.0	171712	2.1	89248	4.0	17337	20.4
E	327279	1.0	138386	2.4	113439	2.9	16529	19.8

Time - Seconds

	Edge Map		STORED		Raw path		Refined path	
	value	i	value	i	value	i	value	i
A	6	1.0	4	1.5	0.83	7.2	-	-
B	1017	1.0	293	3.5	81	12.6	-	-
C	1056	1.0	649	1.6	397	2.7	236	4.5
D	5293	1.0	2067	2.6	975	5.4	208	25.4
E	4835	1.0	1382	3.5	1209	4.0	202	23.9

Conclusion

- Index Fabric combines the advantages of both Patricia tries and B-trees
  - Scaling property of Patricia tries
  - Balanced and optimized for disk-based access like B-trees
- Index Fabric can be built over relational database
  - Leverage the maturity of relational database
- Not need a priori knowledge of the schema of data
  - Can handle XML data without DTD

To Think About

- Need DBA to decide which refined path is appropriate
  - Why not build refines paths by some learning policies which can adapt to the query patterns efficiently
- Can not process queries with wildcards in path expressions efficiently
  - How about T-index

To Think About

- The experiments are not persuadable enough
  - Why not compare the performance with some other novel indexes, such as Join index, indexes defined in Lore

The End