Distributed Access Structures

Tree-based techniques

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

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

April 23, 2013

Indexing structures

We assume a (very) large collection C of pairs (k, v), where k is a key and v is the value of an object (seen as row data).

An index on C is a structure that associates the key with the (physical) address of v. It supports *dictionary operations*:

1. insertion *insert*(k, v),
2. deletion *delete*(k),
3. key search *search*(k)*:* v.
4. (optional) range search *range*(k_1, k_2)*:* $v\}$.

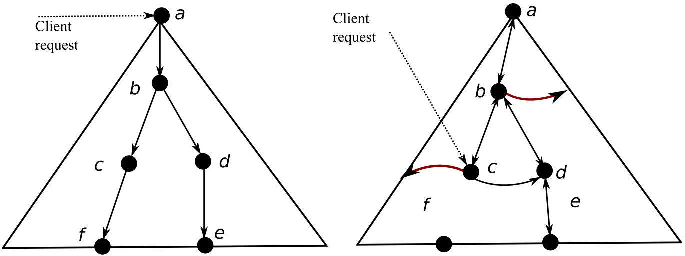
The efficiency of an index is expressed as the number of unit costs required to execute an operation.

NB: in a distributed index, one should also consider (node) *leave* and (node) *join* operations.

Outline

Issues with search trees distribution

All operations follow a top-down path → potential factor of non-scalability



Standard tree *With local routing nodes*

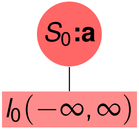
Solutions for distributed structures:

1. *caching* of the tree structure on the the Client node
2. *replication* of parts of the tree
3. *routing tables*, stored at each node, enabling horizontal navigation in the tree.

Case study 1: BATON (P2P)

Conceptually: a standard binary search tree.

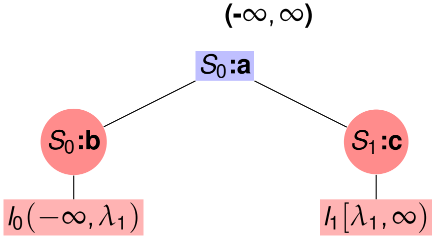
each node covers a range and contains all objects whose key belongs to the range.



Case study 1: BATON (P2P)

Conceptually: a standard binary search tree.

When a server is added, a split occurs, and objects are evenly distributed.
A split generates a routing node and a data node – They can be allocated to a same server.
The range of a routing node covers its subtree.



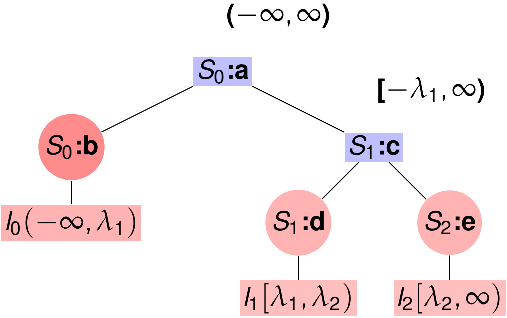
Case study 1: BATON (P2P)

Conceptually: a standard binary search tree.

The tree grows by splitting leaves and adding a local routing node.

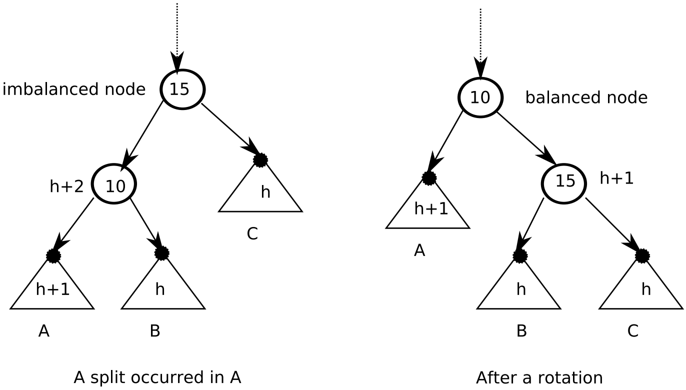
The tree is balanced iff, at each node, the subtrees heights do not differ by more than 1 (e.g., AVL trees).

With non-uniform datasets, split may lead to imbalance.



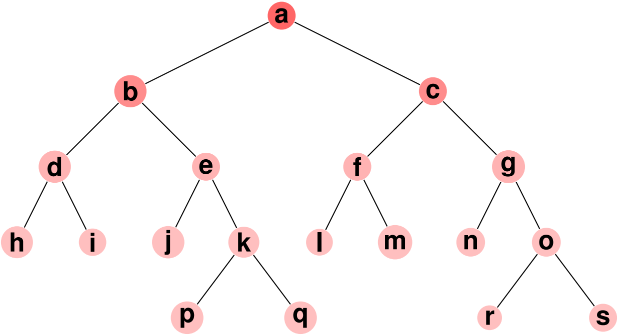
Balancing the tree

When the tree gets imbalanced, a *rotation* is required (still similar to AVL trees):



The approach is still non scalable – every path goes through the root.

A complete example



If we do not add some information: node a receives all the messages, node b receives half of the messages, node d 1/4 of the messages, etc.

⇒ we will partially replicate the tree structure at each node to balance the query load.

Routing tables

Each node stores *routing tables*, that consist of:

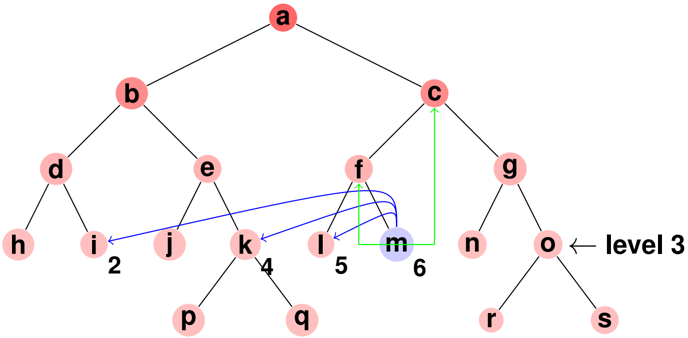
1. parent, left child and right child addresses;
2. previous and next adjacent nodes in in-order traversal;
3. left and right routing tables, that reference nodes at the same level and at position $pos + / - 2^i, i = 0, 1, 2, \ldots$.

Ideas

1. the amount of replication is limited (each node knows a number of “friends" which is logarithmic in the total number of nodes)
2. each node knows better the nodes which are close, than nodes which are far.

Routing tables: example

The left routing table (blue edges) refers to nodes at respective positions $6 - 2^0 = 5$, $6 - 2^1 = 4$, and $6 - 2^2 = 2$.



Note that the gap between two friends f_i and f_{i+1} gets larger as i increases ($2^{i+1} - 2^i = 2^i$).

The number of friends is $\log N$, N being the number of nodes in the considered level.

The routing table of node m

Node m must maintain the following information

Node m – level: 3 – pos: 6 Parent: f – Lchild: null – Rchild: null				
Left adj.: f – Right adj.: c				
Left routing table				
i	node	left	right	range
0	l	null	null	$[l_{min}, l_{max}]$
1	k	p	q	$[k_{min}, k_{max}]$
2	i	null	null	$[i_{min}, i_{max}]$
Right routing table				
i	node	left	right	range
0	n	null	null	$[n_{min}, n_{max}]$
1	o	s	t	$[o_{min}, o_{max}]$

$\Rightarrow$ heavy work when something changes in the network.

Search operations

A *search(k)* request is sent by a Client node to any peer p in the structure. Two steps:

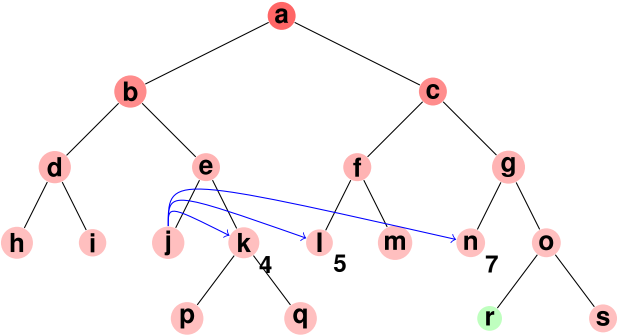
- (horizontal) p looks in its routing table for a node p' at the same level that covers k
  $\rightarrow$ p' is not a friend of p? then there is a friend of p that knows p' better than p.
- (top-down) from p', a standard top-down path is followed.

Procedure: p chooses its farthest friends p'' whose lower bound is smaller than k

Search space halved at each step $\Rightarrow$ ensures that p' is found after at most $\log N$ iterations.

Example of search

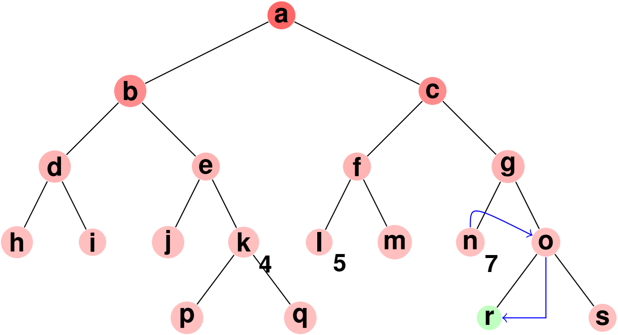
Assume a request sent to node *j* for a key that belongs to node *r*



Blue edges: the (right) friends of *j*; so *j* must forward the request to *n*, its farthest friends whose lower bound is smaller than *k*.

Example of search

Now *n* looks in its own routing table to forward the search.



n knows this part of the tree better than *j*: it finds *o*, the ancestor of *r*, and a downward path is then initiated.

Outline

Case study 2: Bigtable

Can be seen as a distributed *map* structure, with features taken from B-trees, and from non-dense indexed files.

Context: very different from Baton.

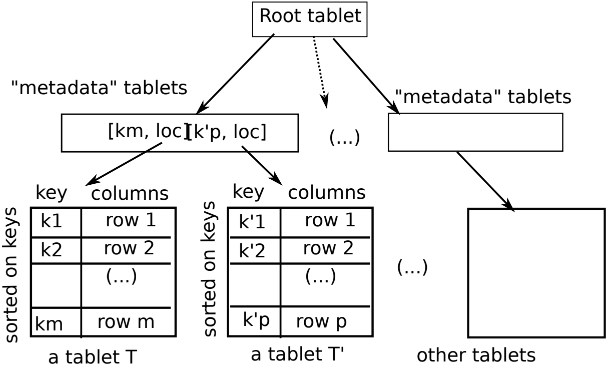
- a controlled environment, with homogeneous servers located in a Data Center;
- a stable organization, with long-term storage of large structured data;
- a data model (column-oriented tables with versioning)

Design: very different as well

- close to e B-tree, with large capacity leaves
- scalability is achieved by a cache maintained by Client nodes.

Overview of Bigtable structure

Leaf level: a “table” organized in “rows” indexed by a key. Rows are stored in lexicographic order on the key values.

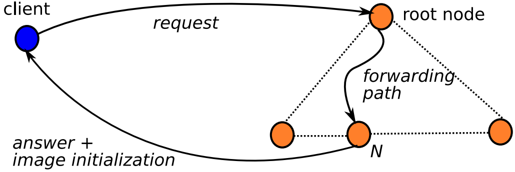


The table is partitioned in “tablets”, and tablets are indexed by upper levels.

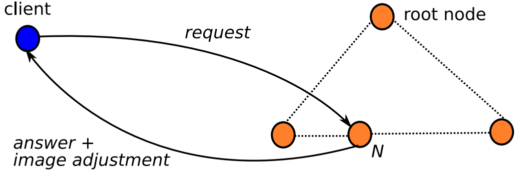
Full tablets are split, with upward adjustment.

Architecture: one Master - many Servers

The Master maintains the root node and carries out administrative tasks.



a) A new client contacts a distributed system

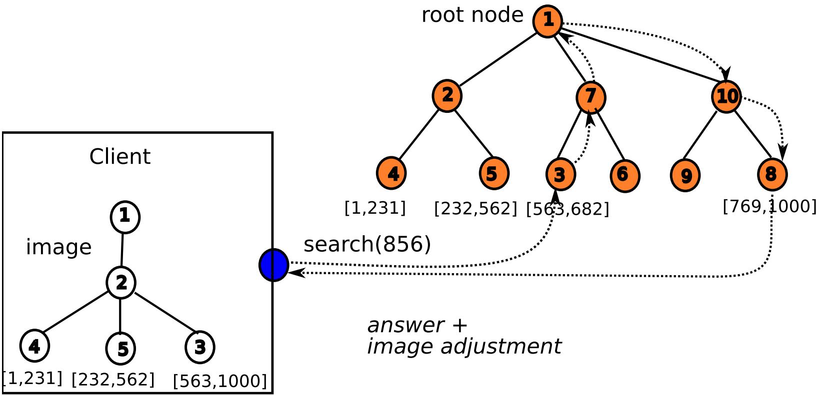


b) Using its image, the client directly contacts N

Scalability is obtained with Client cache that stores a (possibly outdated) image of the tree.

Example of an out-of-range request followed by an adjutment

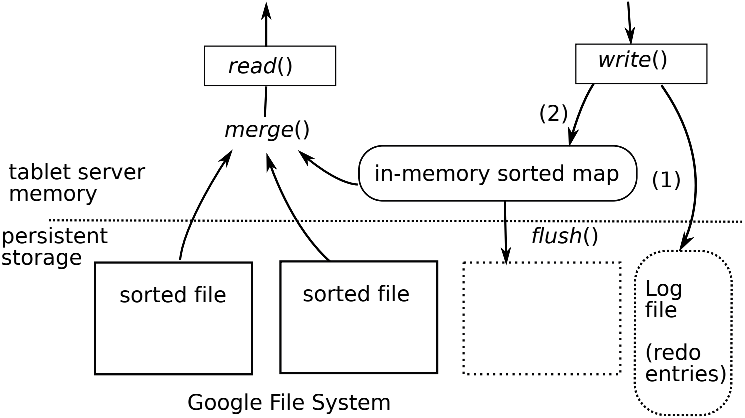
A Client request may fail, due to an out-of-date image of the tree.



An adjustment requires at most *height(Tree)* rounds of messages.

Persistence management in Bigtable

Problem: how can we maintain the sorted structure of tablets?



Distributed indexing: what you should remember

Key point: Scalability. No single point of failure; even load distribution over all the nodes. Technical means:

- Distribute (and maintain) routing information.
  $\Rightarrow$ trade-off between maintenance cost and operations cost.
- Cache an image of the structure (e.g., in the Client).
  $\Rightarrow$ design a convergence protocol if the image gets outdated.

Key point: efficiency. Clearly depends on the amount of information replicated at each node or at the Cient.

- Stable systems: the structure can be duplicated at each node. Allows $O(1)$ cost – low maintenance.
- Highly dynamic systems: very hard to maintain a consistent view of the structure for each participant.

Always: be ready to face a failure somewhere; detect failures, use and replication and deal with it.