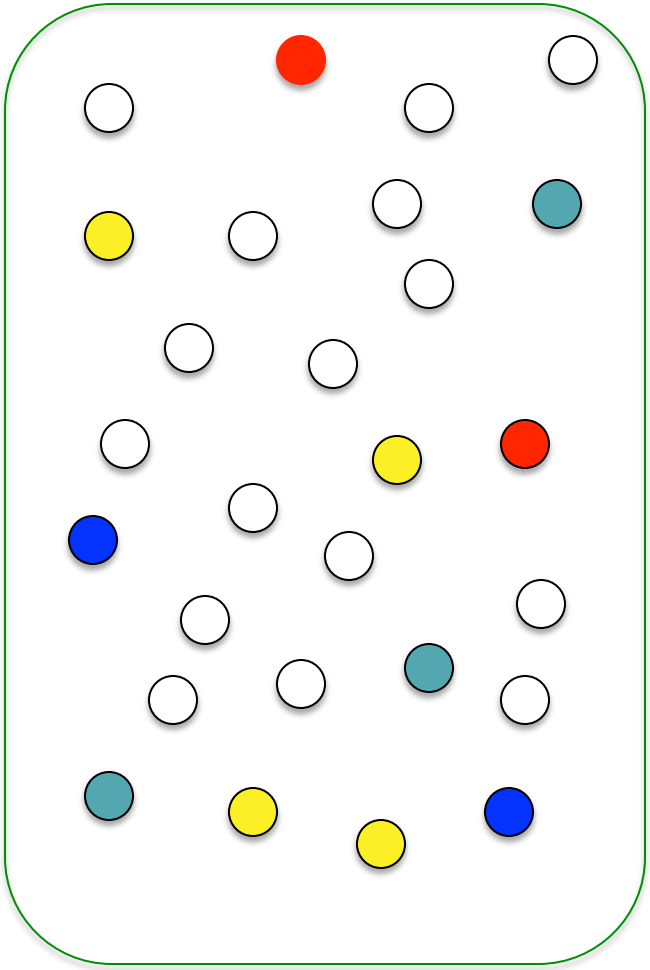
Accessing nearby copies of replicated objects

Greg Plaxton,
Rajmohan Rajaraman,
Andrea Richa

SPAA 1997

Goal

A set A of m *shared objects* resides in a network G. Design an algorithm of so that processes can access a nearby copy of it. Supported operations are *insert, delete, read* (but not *write*) on shared objects.



Challenge

What is the challenge here? Think about this:

1. if every node maintains a copy, then read is fast, but insert or delete are very expensive. Also, storage overhead grows.
2. If the object is stored in an arbitrary node, then the access may take a long time.

The algorithm must be efficient w.r.t. both time and space

Plaxton Routing

Plaxton routing provides an answer: it has been used in several important P2P networks like Microsoft's **Pastry** and UC Berkeley's **Tapestry** *(Zhao, Kubiatowicz, Joseph et al.) that is also the backbone of* **Oceanstore,** a persistent global-scale storage system

Object and Node names

Objects and nodes acquire **names** independent of their location and semantic properties, in the form of random fixed-length bit-sequences (160 bits) using a hashing algorithm such as SHA-1 . This leads to roughly *even distribution* of objects in the name space)

The Main Idea

For every object, form a rooted tree. The root node stores a pointer to the **server** that stores the object. The root does not hold a copy of the object itself. Once you access the root, you can navigate to the server storing that object.

How to choose a root for a given object?

The Main idea continued

Embed an n-node virtual height-balanced tree T into the network. Each node **u** maintains information about copies of the object in the subtree of T rooted at **u**.

Choosing the root

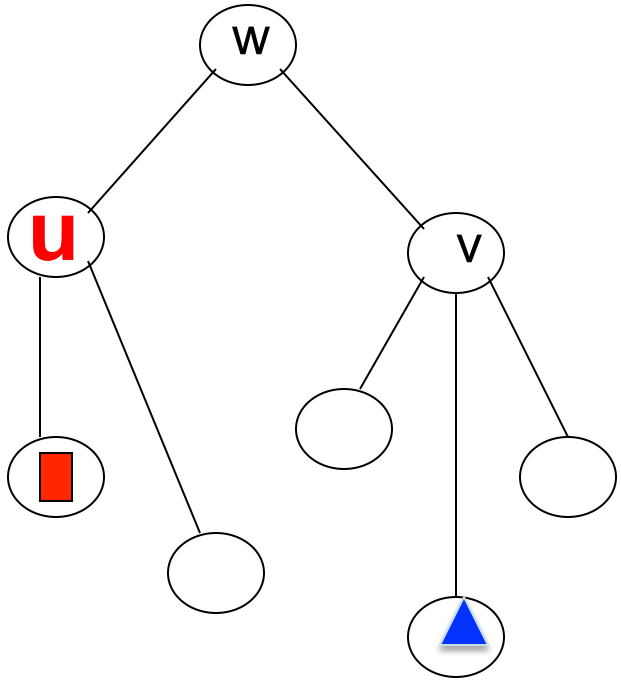
An object's root is a node whose **name** matches the object's **name** *in the largest number of trailing bit positions*. In case there are multiple such nodes, choose the node with the largest such id.

[Life of Pi]:	010 010110101000
Node X:	110 010110101000
Node Y:	011 010110101000

"Life of Pi" will be mapped to Node X

Example of search

To access a copy, **u** searches the subtree under it. If the copy or a pointer to the object is available, then **u** uses that information, otherwise, it passes the request to its parent.



Plaxton tree

Plaxton tree (some call it Plaxton mesh) is a data structure that allows peers to efficiently locate objects, and route to them across an arbitrarily-sized network, using a small routing map at each hop. Each node serves as a client, server and a router.

Examples of Object names

Represent node ids and object names as strings of digits using base = 2^b, where b is a fixed positive integer. Let b=2. Thus, the name of object B = 1032 (base 4 = 2^b), name of object C = 3011, name of a node u = 3122 and so on.

Neighbor map

Consider an example with b=3, i.e. the id base $2^b = 8$. Level i entries match i suffix entries. Number of entries per level = ID base = 8

Each entry is the suffix of a matching node with least cost. If no such entry exists, then pick the one that has highest id & largest suffix match

These are all **primary entries**

L3	L2	L1	L0
0642	x042	xx02	xxx0
1642	x142	xx12	xxx1
2642	x242	xx22	xxx2
3642	x342	xx32	xxx3
4642	x442	xx42	xxx4
5642	x542	xx52	xxx5
6642	x642	xx62	xxx6
7642	x742	xx72	xxx7

Neighbor map of node 5642

More on neighbor map

In addition to primary entries (lowest access cost), each level contains secondary entries.

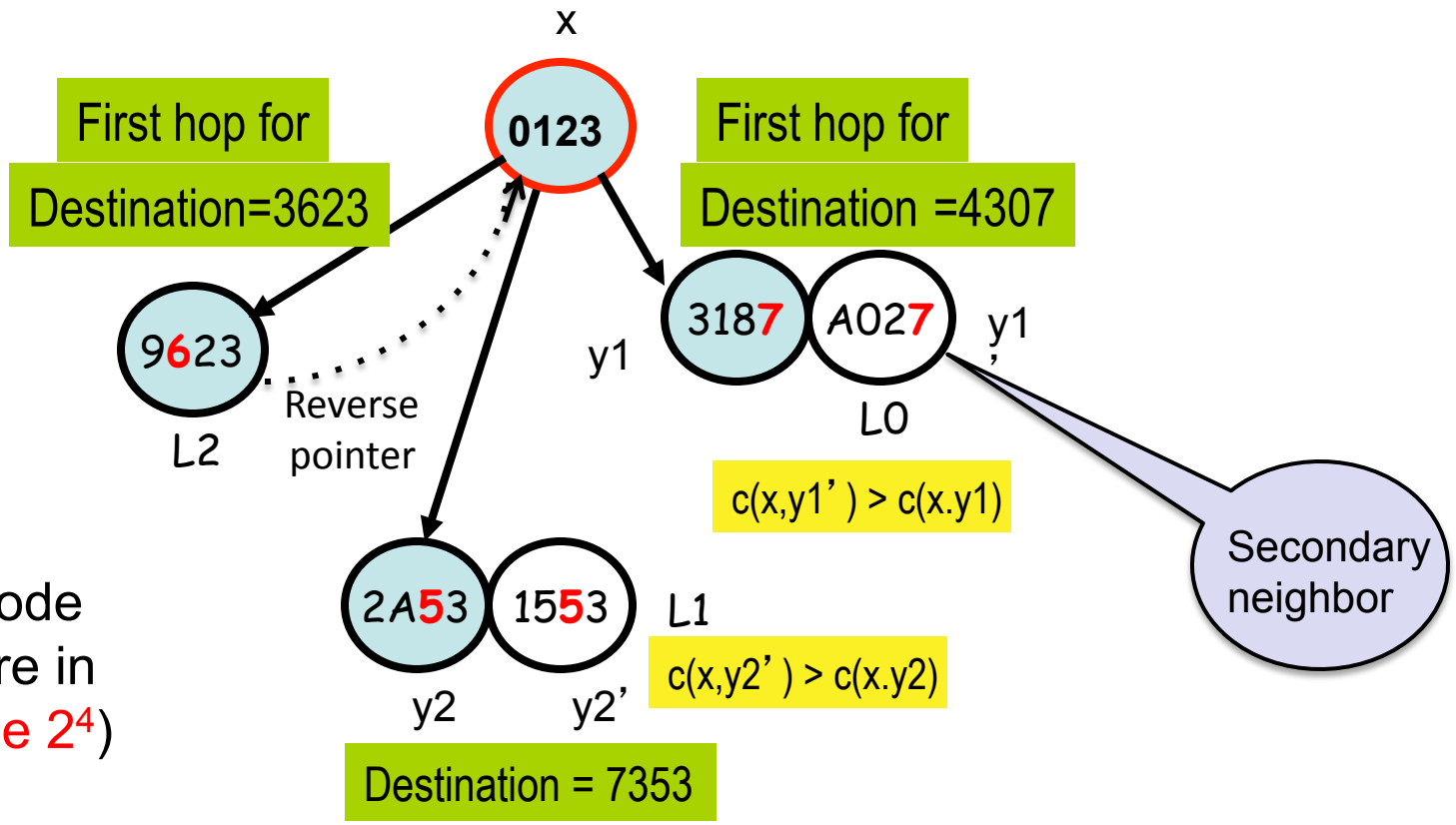
-- A node **u** is a secondary entry of node x at a level i, if (1) it is not the primary neighbor, and (2) **the cost c(x,u) ≤ c(x,w)** for all non-primary neighbor w with a matching suffix of size i.

-- Each node also stores reverse neighbors for each level. A node y is a reverse neighbor of x if and only if x is a primary neighbor of y.

-- Each node stores a pointer list (O,y,k) for each object in its subtree, where O is the object, y is the node holding a copy of A and k is the upper bound of the cost to access A

All entries are statically chosen, and this needs global knowledge (-)

Neighbor map: another example



b=4,so node names are in Hex (base 2^4)

Size of the routing table: base * log $_{base}$ (n)

Routing Algorithm

To route to root of an object

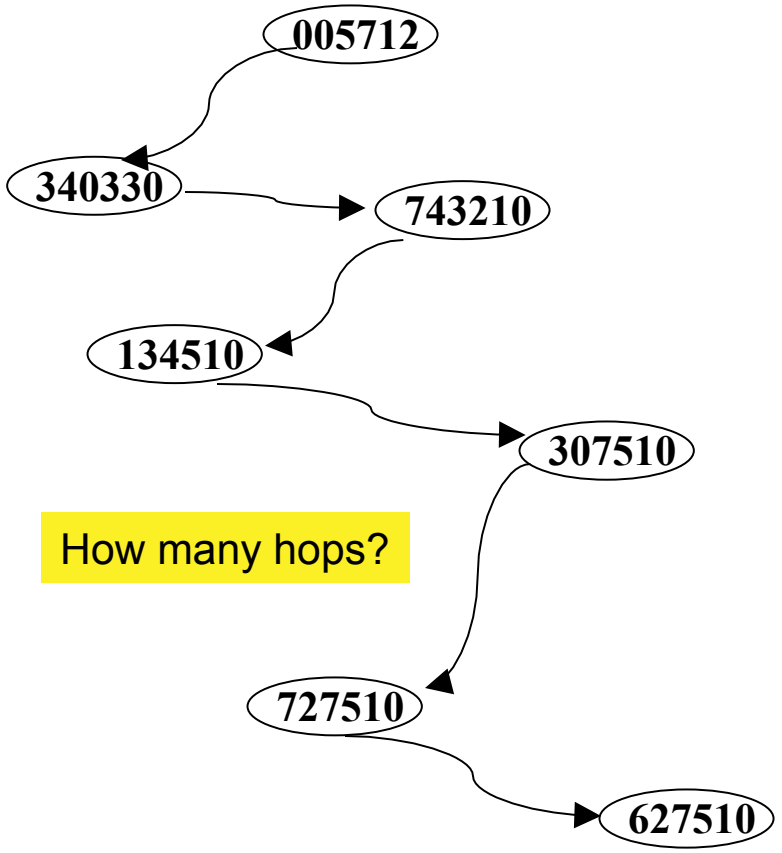
- Let shared suffix (between root id and node id) have length s
- If s = $\log_{base}(n)$ (i.e. this node is the root) then done, else look at level s+1
- Match the next digit of the destination id
- Send the message to that node

Eventually the message gets relayed to the destination

This is called **suffix routing** (one could also use prefix routing).

Example of suffix routing

Consider routing a message from 005712 → 627510 (base = 8)



How many hops?

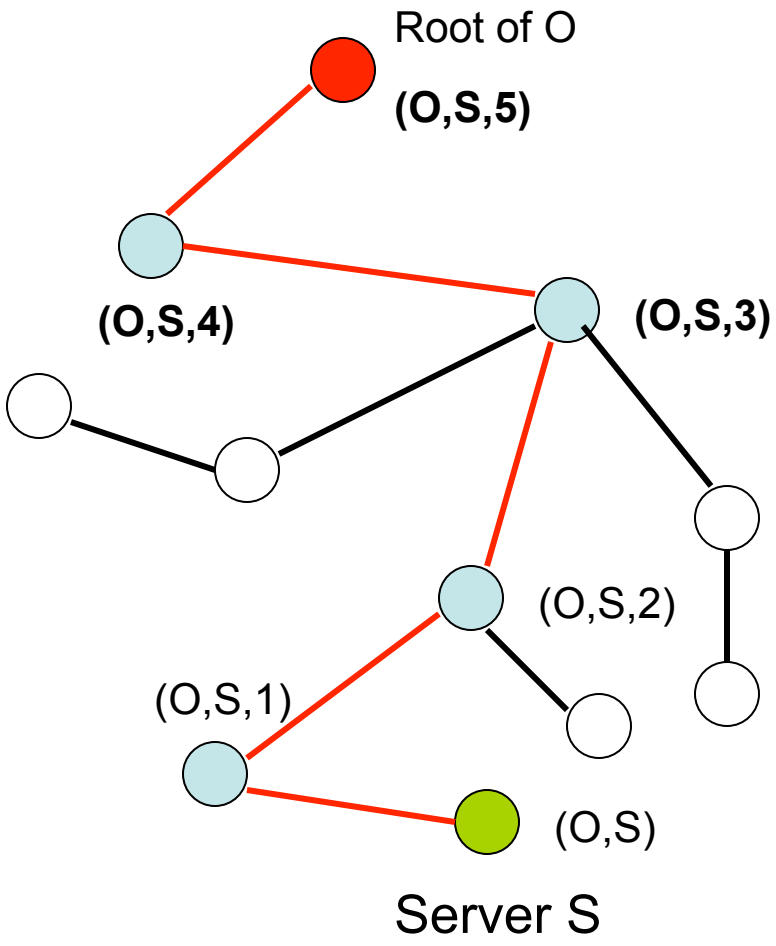
Pointer list

Each node x has a ***pointer list P(x):***

P(x) is a list of triples **(O, S, k),**

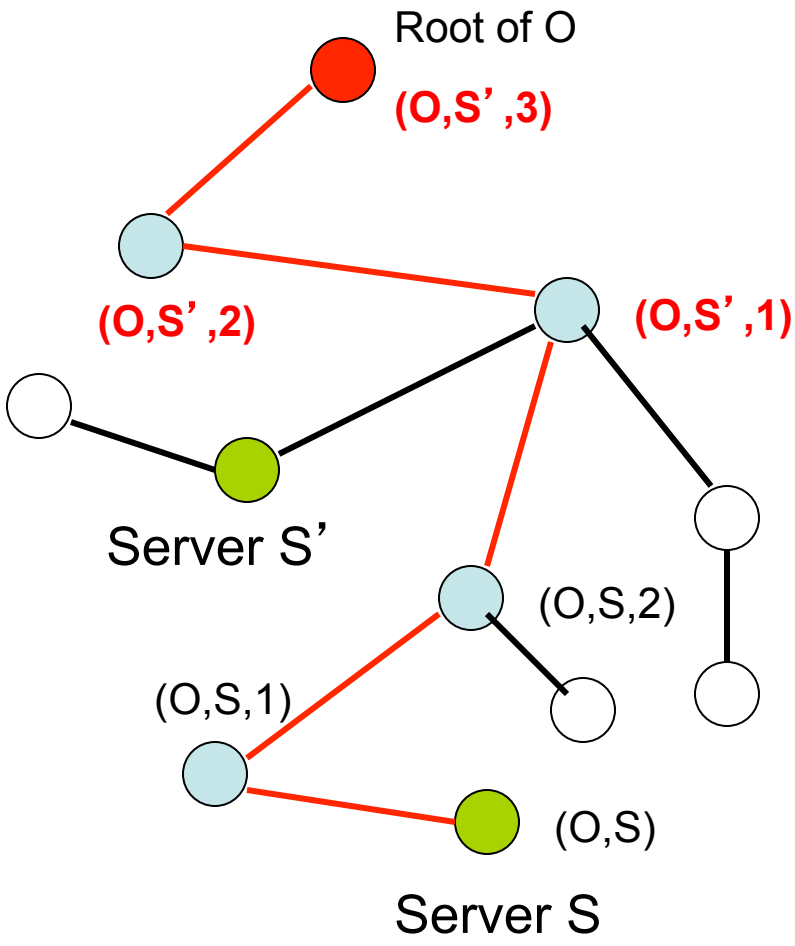
- where O is the object,
- Node S holds a copy of O,
- and k is the max cost of the access.

The pointer list is updated by ***insert*** and ***delete*** operations.



Inserting a copy

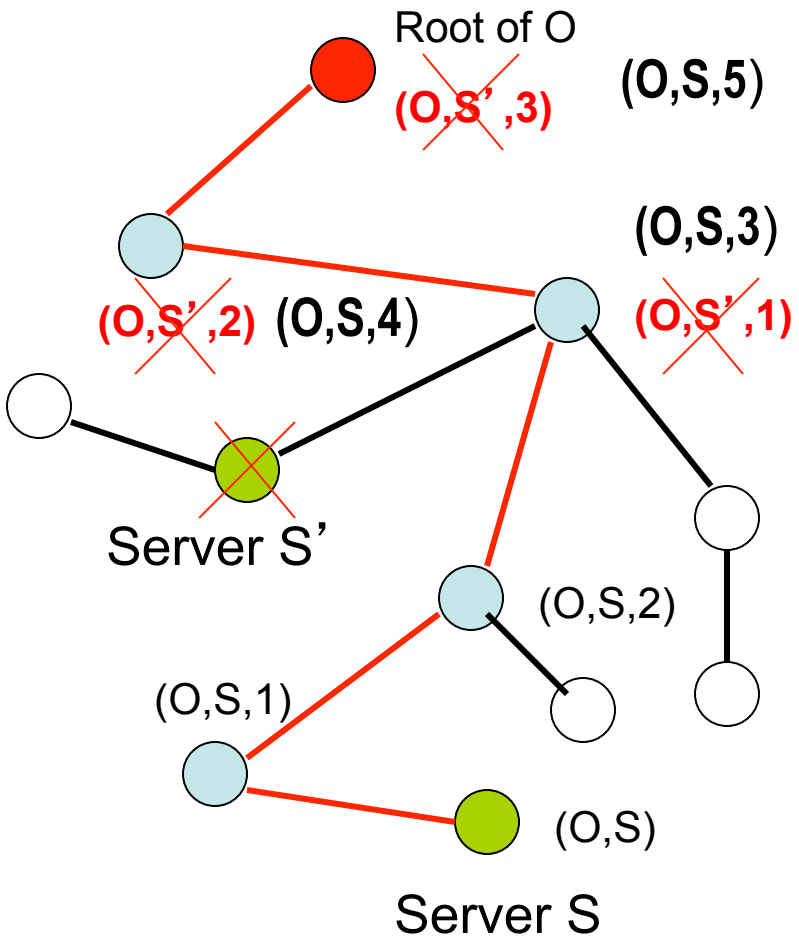
Intermediate nodes maintain the minimum cost of access. While inserting a duplicate copy, the pointers are modified, so that they direct to the closest copy



Deleting a copy

Removes all pointers to that copy of the object in the pointer list on the nodes leading to the root

Otherwise, it uses the reverse pointers to update the entry.



Results

Let C= (max{c(u,v): u,v in V}

Cost of reading a file A of length L(A) from node v by node u =f(L(A)).c(u,v)

Cost of inserting a new object = O(C)

Cost of deleting an existing object = O(C log n)

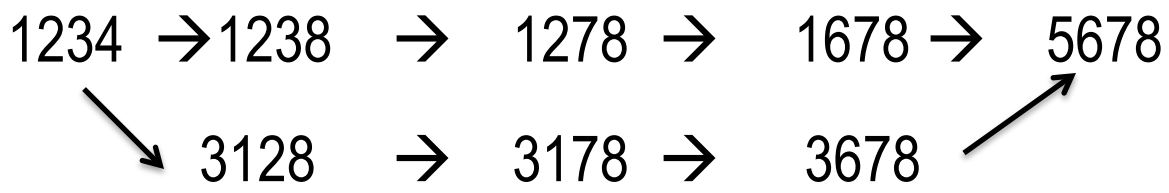
Benefits and Limitations

+ Scalable solution

  Small routing table. All routing done using locally available data

+ Existing objects are guaranteed to be found

+ Simple fault handling



+ Optimal routing distance

Benefits and limitations

- Needs global knowledge to form the neighbor tables. How can we solve it?

- The root node for each object may be a possible bottleneck.