Shared Objects

Thursday, June 26, 2014
10:16 AM

common resource / object

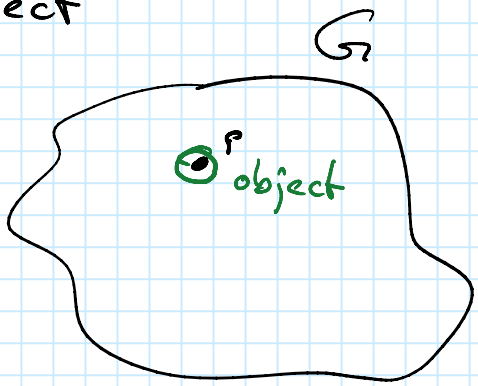
guarantee mutual exclusion of accesses

Centralized Solution

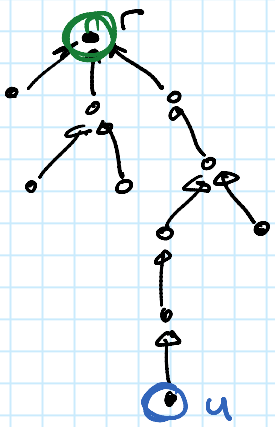
root node r has the object

node u wants to access obj.

u sends request to r
r sends back answer



assume a spanning tree rooted at r



all accesses by u

↳ would be better to store obj. at node u

Home-Based Solution

obj. has a home base r

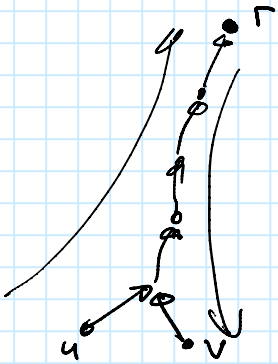
u wants to access obj.

↳ if u has obj. ✓

otherwise: asks r where obj. is

→ object is moved to u

Problem: triangular routing problem

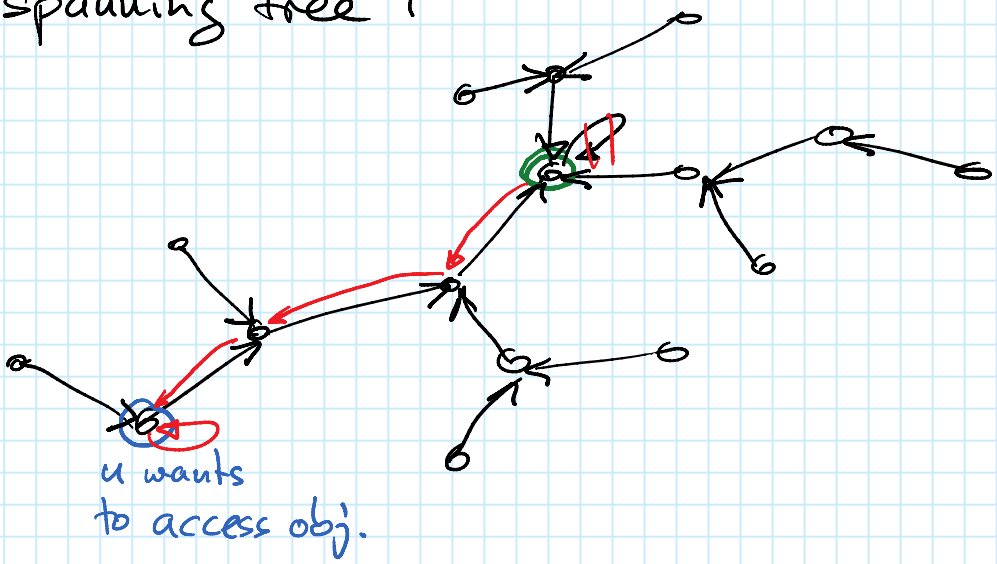


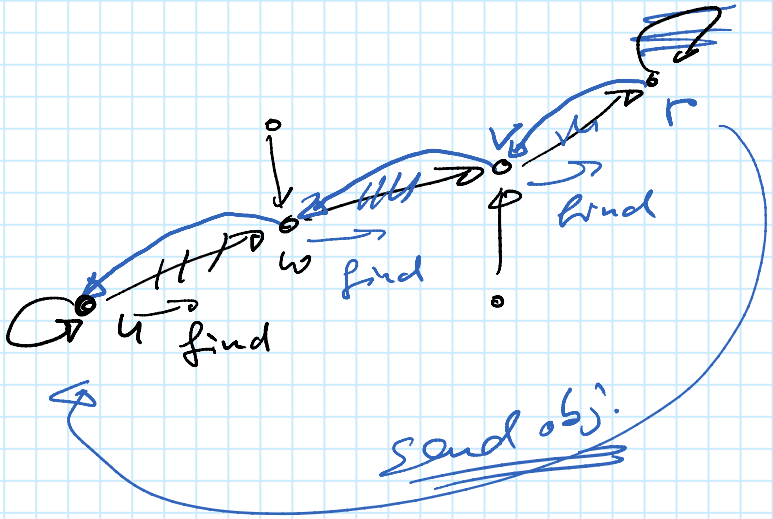
u and v access obj. alternatingly

large overhead for searching object

Arrow Protocol

assume spanning tree T





Algorithm 4.3 Shared Object: Arrow Algorithm

Initialization: As for Algorithm 4.1, we are given a rooted spanning tree. Each node has a pointer to its parent, the root r is its own parent. The variable is initially stored at r. For all nodes v, v.successor := **null**, v.wait := **false**.

Start Find Request at Node u:

1: **do atomically**
2: u sends "find by u" message to parent node
3: u.parent := u
4: u.wait := **true**
5: **end do**

Upon w Receiving "Find by u" Message from Node v:

6: **do atomically**
7: **if** w.parent $\neq w$ **then**
8: w sends "find by u" message to parent
9: w.parent := v
10: **else**
11: w.parent := v
12: **if not** w.wait **then**
13: send variable to u // *w holds var. but does not need it any more*
14: **else**
15: w.successor := u // *w will send variable to u a.s.a.p.*
16: **end if**
17: **end if**
18: **end do**

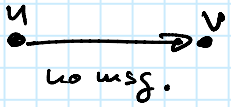
Upon w Receiving Shared Object:

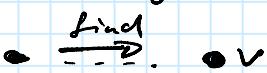
19: perform operation on shared object
20: **do atomically**
21: w.wait := **false**
22: **if** w.successor $\neq$ **null then**
23: send variable to w.successor
24: w.successor := **null**
25: **end if**
26: **end do**

Protocol works in a completely asynchronous and concurrent setting.

Theorem: A "find" request terminates with time and message complexity D (D: diam. of T)

Lemma:
An edge $\{u,v\}$ of T is one of 4 states

1.) arrow from u to v (no msg.) 

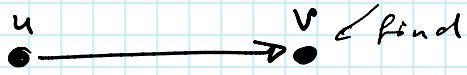
2.) message from u to v (no arrow) 

3.) arrow from v to u (no msg.)

4.) msg. from v to u (no arrow)

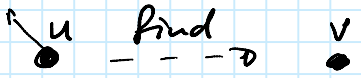
State i is always followed by state $i+1$ $(4+1=1)$

Proof: assume, initially $\{u,v\}$ is in state 1

State 1: 

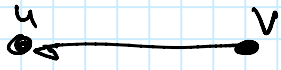
find req. at node u
↳ u sends find msg. to v } atom.
u redir. pointer

⇓

State 2: 

msg. reaches node v
v redirects pointer to u

⇓

State 3: 

⟹ State 4 ⟹ State 1

□

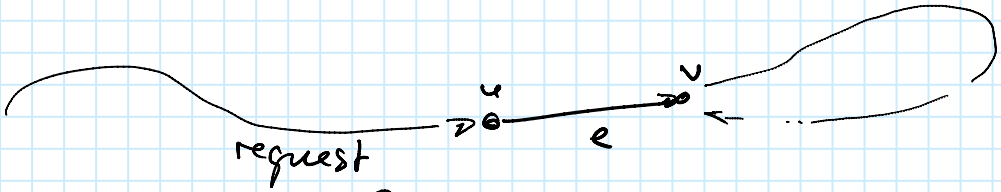
Proof of theorem:

msg. travels on static tree

Suff. to show that no edge is traversed 2x

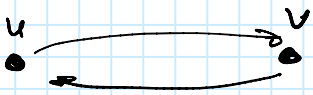
Consider first req. which traverses some edge twice

$\Rightarrow$ $e = \{u, v\}$ is the first edge traversed twice



e is the first edge

$\Rightarrow$ traverse e twice without traversing any other edges in between



by Lemma:

when reaching v : edge is in state 2

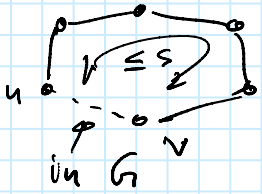
parent of v has to be u $\Rightarrow$ edge is state 3

$\Rightarrow$ contradiction to Lemma.

□

Remarks

- Performance depends on properties of T
  ↳ ideally: tree with small diameter, small stretch s

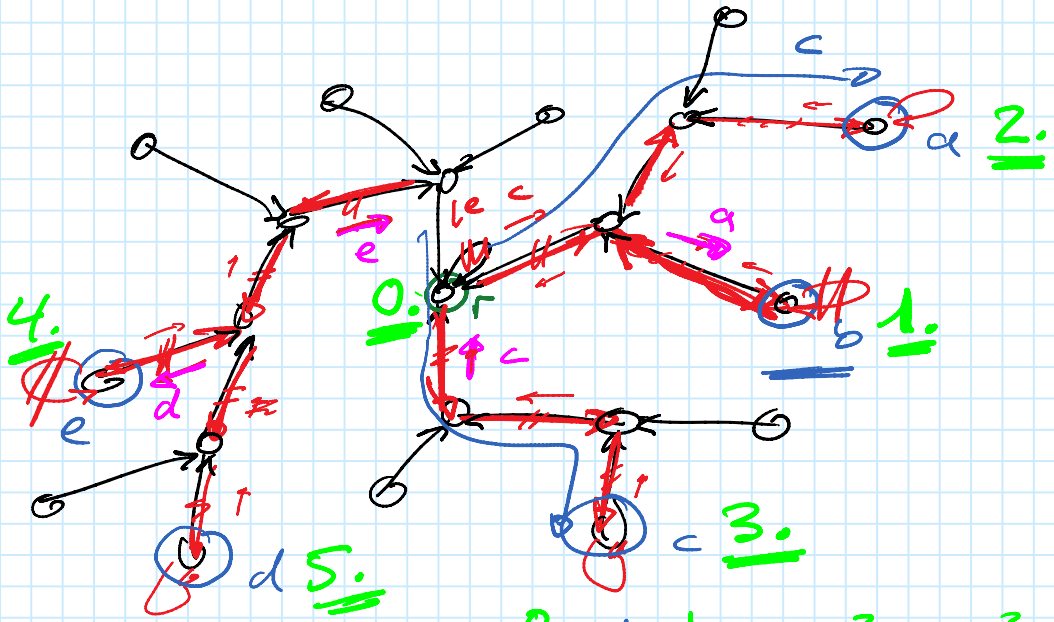


$$\text{Stretch } s := \max_{u,v} \frac{d_T(u,v)}{d_G(u,v)}$$

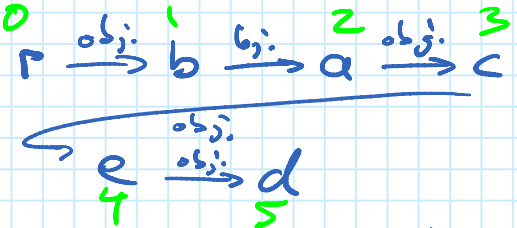
- algorithm seems to be particularly good if there is a lot of concurrency

Concurrent Analysis

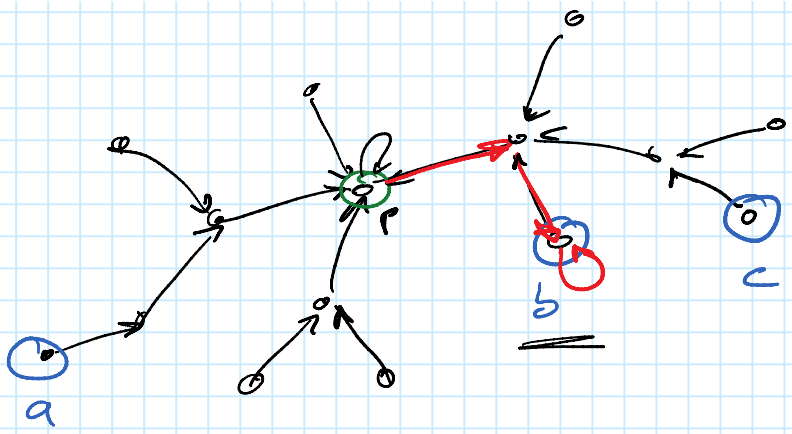
- assume synchronous model
- "one-shot" execution
  set $S \subseteq V$ that wants to access obj.
  all requests start at time 0.



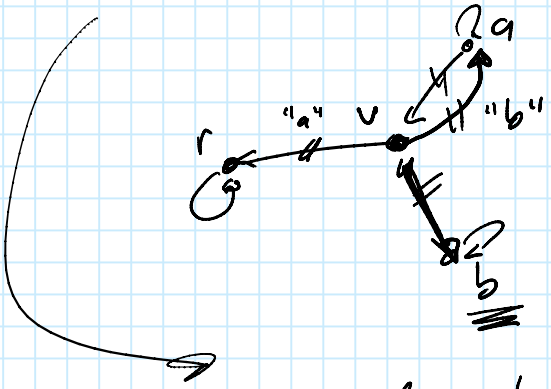
b gets obj. from r
a finds b
d finds e
c finds a
e finds c



d is the new root



b reaches r (b is closest to r)



c reaches b

a reaches c

Theorem:

Total cost is $O(\log|S| \cdot m^*)$, where m^* is the optimal msg. complexity on T

assume we know S, can order the req. in an optimal way.

optimal ordering

min. the total #messages

cost of an ordering

$H=(S, E)$

cost of a TSP tour on T for given ordering (path)

Arrow ordering: greedy TSP tour/path (nearest-neighbor)

metric TSP: greedy is a $O(\log|S|)$-approx.