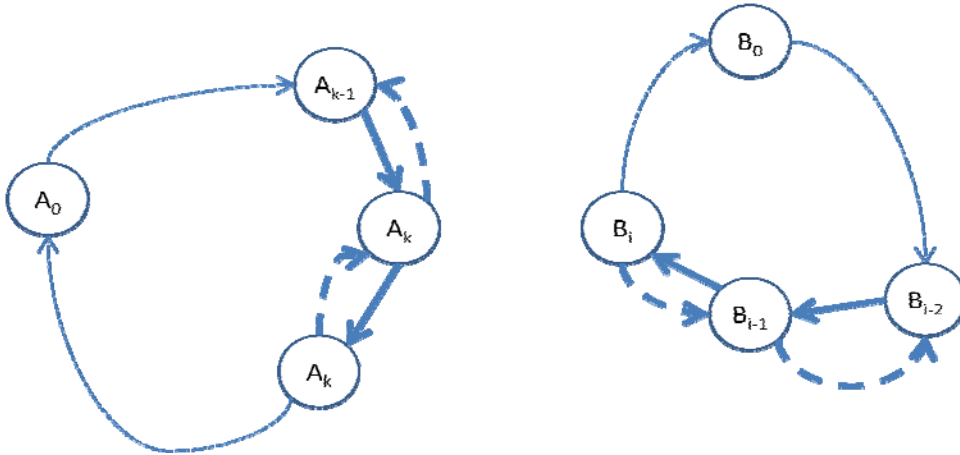


CS138 Homework Assignment 1 Solutions

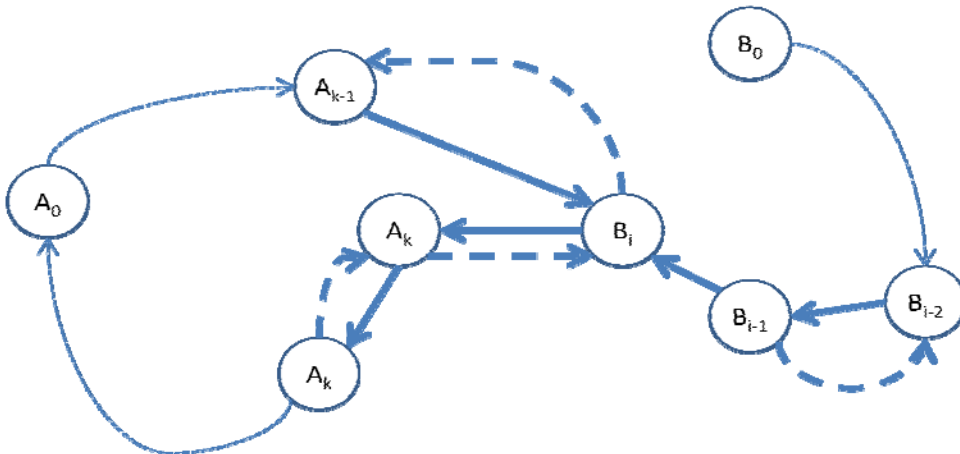
Spring 2009

- Suppose we have two disjoint Chord rings, labeled A and B. Assuming that one node in ring B knows the address of a node in ring A, give an algorithm for combining the two rings into a single well formed Chord ring. You may assume that no nodes are added, fail, or are removed while the combining is taking place. Give a good argument for why your algorithm works. (Hint: use the stabilization procedure given in slide II-30.)

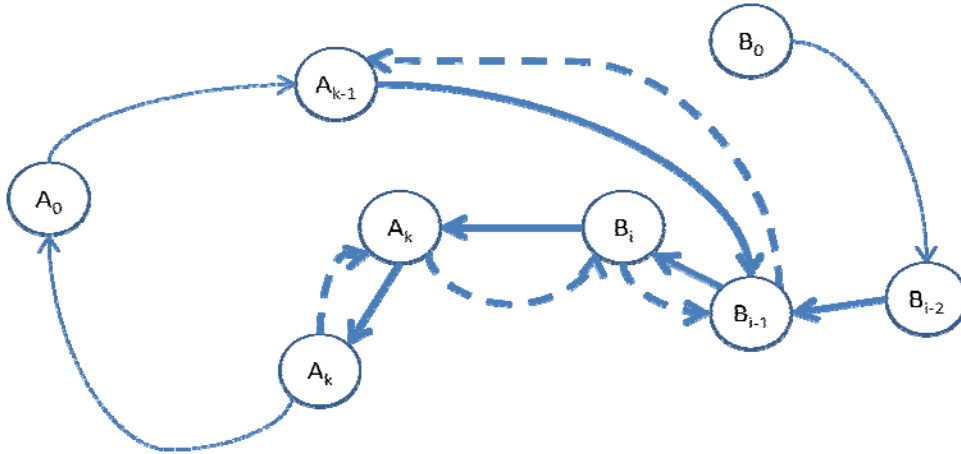
Consider the two rings sketched below. Successor links are shown by thick solid lines and predecessor links by heavy dashed lines. Nodes not shown are indicated by thin dotted lines.



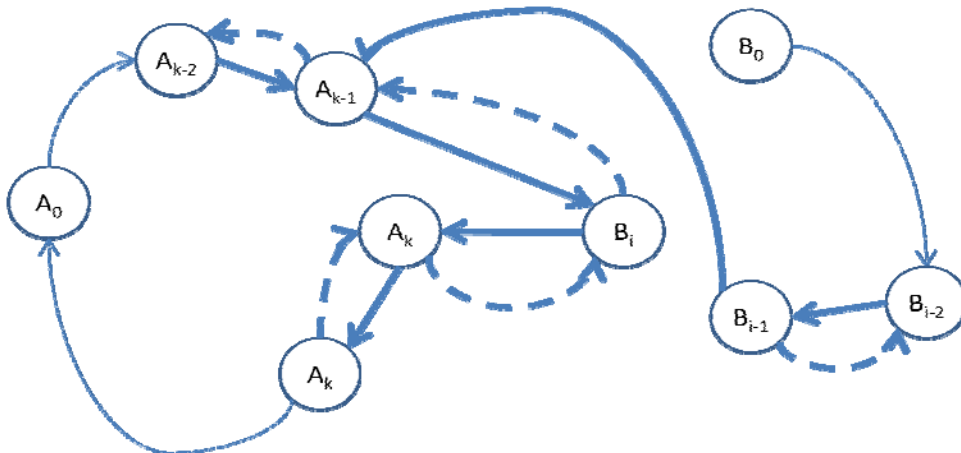
Suppose node B_i knows the address of some node A_j . B_i then sets its successor to the result of calling $A_j.successor(B_i)$, which, let's say, is A_k . Thus the B ring is now "broken" in that B_i points to its successor in the A ring rather than to its successor in the B ring. Assuming no additions or deletions to either ring, if any node other than B_i calls *stabilize*, no changes will occur. So, suppose B_i calls *stabilize*. Since A_k is B_i 's successor in ring A, B_i is greater than A_k 's predecessor and thus the call to *stabilize* sets A_k 's predecessor to B_i . Now, when A_k 's predecessor calls *stabilize*, it will find that its successor's predecessor (B_i) should lie between it and A_k , so it sets its successor to be B_i and calls $B_i.notify(A_{k-1})$, which sets B_i 's predecessor to be A_{k-1} . The result looks something like the following:



At this point, B_i is effectively part of the A ring and B_{i-1} is the next node of the B ring that is to be inserted into the A ring. We would like to set B_{i-1} 's successor to be its successor in the A ring. This will happen after some number of calls to *stabilize*, as follows: B_{i-1} calls *stabilize* and examines its successor's predecessor — A_{k-1} . If B_{i-1} is greater than A_{k-1} , the B_i is B_{i-1} 's successor in ring A and, after a call to notify, B_i 's predecessor is set to B_{i-1} . A_{k-1} will at some point call *stabilize* and then notify, setting its successor to B_{i-1} and B_{i-1} 's predecessor to A_{k-1} :



Alternatively, if B_{i-1} is less than A_{k-1} , then B_{i-1} 's successor is set to point to A_{k-1} :



This process will continue until B_{i-1} 's successor link actually points to its successor in ring A. The entire procedure is repeated with B_{i-2} , B_{i-3} , etc., until all B nodes have been inserted into ring A.

Now consider the objects stored at the nodes. Whenever a node x changes its successor, some or all of the objects in the target of its new successor, y , may have hashes that are less than x . These objects should then be moved to x . If x is in ring B and y is in ring A, the effect will be that an object previously found in ring A will now be found only in ring B. However, x will soon be in ring A. Thus it may be necessary to repeat failed searches.

2. Suppose a node is to be deleted in Tapestry (it didn't just temporarily fail — it was sold). Explain what must be done to remove it completely, without losing any important information or data.

Tapestry nodes hold three sorts of information: they may be the root nodes for objects (perhaps as a surrogate) and thus hold object-location information, they hold objects, and they hold cached

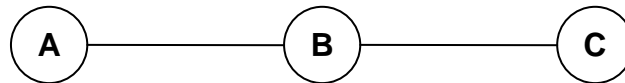
object-location information (if they are on the publication path between a node holding an object and the object's root node). Tapestry nodes also appear in routing tables.

Suppose that the node being removed is a root node for one or more objects. The node's removal will cause the object-location information to be lost, but recall that object holders periodically republish their objects. Thus a new root will be established soon — no information is lost simply by deleting node's object-location information. Similarly, any cached object-location information will be replaced once the object holder republishes the object.

The routing tables of neighboring nodes refer to the node being deleted. However, this information will be automatically removed once a neighboring node attempts to contact the node and times out. A node might contact its neighbors via its backpointers and inform them that it is about to disappear, but this would be just an optimization and not necessary.

If a node is holding an object, then the object would be lost if the node were removed without doing anything about it. If this is the only copy of the object, then information would be lost. Thus the node should transfer its objects to other nodes. It would make sense to transfer them to nearby nodes, so as to retain any locality. There will be a transient problem in that searches for these objects might fail because old object-location information still refers to the node being removed. This could be mitigated by having the node send node-removal messages to its objects roots, informing them (and all nodes along the path) that it no longer holds its objects.

3. [From Peterson and Davie, Computer Networks.] Consider the following network in which a distance-vector routing protocol is used and both links have cost one:



Suppose the link between A and B fails.

- a. Give a sequence of routing-table updates that leads to a routing loop between B and C. Assume that split horizon is not used.

Immediately after the link fails, C sends B its routing table. C's routing table contains the entry (A, 2, B), where the elements of the tuple are the destination node, the distance to that node, and which node to contact first to route a packet to the given node. Since the third value in the tuple isn't transmitted, node B receives the tuple (A, 2) from node C, and therefore updates its routing table to contain (A, 3, C). When Node B sends its routing table to node C, C updates its routing table to contain (A, 4, B), since it sees that B's distance to A has just increased. And so the cycle continues.

- b. Estimate the probability of the scenario given in part a, assuming B and C send out routing updates at random times, each at the same average rate.

At the instant that B discovers the B-A failure there is a 50% chance that the next report will be from C and a 50% chance that B will issue the next report. If it is B goes, the loop will not form; if C does, it will.

- c. Estimate the probability of a loop forming if B broadcasts an updated report within one second of discovering the B-A link failure, and C broadcasts every 60 seconds.

At the instant B discovers the B-A failure, let t be the time until C's next broadcast. t is equally likely to occur anywhere in the interval $0 \leq t \leq 60$. A loop forms if and only if B broadcasts first, which happens if and only if $t < 1.0$ sec; the probability of this is $1/60$.