CS425/CS525 Final Exam

May 8th, 2008

Write your answers in the blue book(s). Justify your answers. Work alone. Do not use any notes or books.

There are four problems on this exam, each worth 20 points, for a total of 80 points. You have approximately three hours to complete this exam.

1 Message passing without failures (20 points)

Suppose you have an asynchronous message-passing system with a complete communication graph, unique node identities, and no failures. Show that any deterministic atomic shared-memory object can be simulated in this model, or give an example of a shared-memory object that can't be simulated.

Solution

Pick some leader node to implement the object. To execute an operation, send the operation to the leader node, then have the leader carry out the operation (sequentially) on its copy of the object and send the results back.

2 A ring buffer (20 points)

Suppose you are given a *ring buffer object* that consists of $k \geq 1$ memory locations $a[0] \ldots a[k-1]$ with an atomic *shift-and-fetch* operation that takes an argument v and (a) shifts v into the buffer, so that $a[i] \leftarrow a[i+1]$ for each i less than $k-1$ and $a[k-1] \leftarrow v$; and (b) returns a snapshot of the new contents of the array (after the shift).

What is the consensus number of this object as a function of k?

Solution

We can clearly solve consensus for at least k processes: each process calls shift-and-fetch on its input, and returns the first non-null value in the buffer.

So now we want to show that we can't solve consensus for $k+1$ processes. Apply the usual FLP-style argument to get to a bivalent configuration C where each of the $k+1$ processes has a pending operation that leads to a univalent configuration. Let e_0 and e_1 be particular operations leading

to 0-valent and 1-valent configurations, respectively, and let $e_2 \dots e_k$ be the remaining $k-1$ pending operations.

We need to argue first that no two distinct operations e_i and e_j are operations of different objects. Suppose that Ce_i is 0-valent and Ce_j is 1-valent; then if e_i and e_j are on different objects, Ce_ie_j (still 0-valent) is indistinguishable by all processes from Ce_je_i (still 1-valent), a contradiction. Alternatively, if e_i and e_j are both b-valent, there exists some $(1-b)$-valent e_k such that e_i and e_j both operate on the same object as e_k, by the preceding argument. So all of $e_0 \dots e_k$ are operations on the same object.

By the usual argument we know that this object can't be a register. Let's show it can't be a ring buffer either. Consider the configurations $Ce_0e_1 \dots e_k$ and $Ce_1 \dots e_k$. These are indistinguishable to the process carrying out e_k (because its sees only the inputs to e_1 through e_k in its snapshot). So they must have the same valence, a contradiction.

It follows that the consensus number of a k-element ring buffer is exactly k.

3 Leader election on a torus (20 points)

An $n \times n$ torus is a graph consisting of n^2 nodes, where each node (i, j), $0 \leq i, j \leq n-1$, is connected to nodes $(i-1, j), (i+1, j), (i, j-1)$, and $(i, j+1)$, where all computation is done mod n.

Suppose you have an asynchronous message-passing system with a communication graph in the form of an $n \times n$ torus. Suppose further that each node has a unique identifier (some large natural number) but doesn't know the value of n. Give an algorithm for leader election in this model with the best message complexity you can come up with.

Solution

First observe that each row and column of the torus is a bidirectional ring, so we can run e.g. Hirschbirg and Sinclair's $O(n \log n)$-message protocol within each of these rings to find the smallest identifier in the ring. We'll use this to construct the following algorithm:

1. Run Hirschbirg-Sinclair in each row to get a local leader for each row; this takes $n \times O(n \log n) = O(n^2 \log n)$ messages. Use an additional n messages per row to distribute the identifier for the row leader to all nodes and initiate the next stage of the protocol.

2. Run Hirschbirg-Sinclair in each column with each node adopting the row leader identifier as its own. This costs another $O(n^2 \log n)$ messages; at the end, every node knows the minimum identifier of all nodes in the torus.

The total message complexity is $O(n^2 \log n)$. (I suspect this is optimal, but I don't have a proof.)

4 An overlay network (20 points)

A collection of n nodes—in an asynchronous message-passing system with a connected, bidirectional communications graph with $O(1)$ links per node—wish to engage in some strictly legitimate file-sharing. Each node starts with some input pair (k, v), where k is a key and v is a value, and the search problem is to find the value v corresponding to a particular key k.

1. Suppose that we can't do any preparation ahead of time. Give an algorithm for searching with the smallest asymptotic worst-case message complexity you can find as a function of n. You may assume that there are no limits on time complexity, message size, or storage space at each node.

2. Suppose now that some designated leader node can initiate a protocol ahead of time to pre-process the data in the nodes before any query is initiated. Give a pre-processing algorithm (that does not depend on which key is eventually searched for) and associated search algorithm such that the search algorithm minimizes the asymptotic worst-case message complexity. Here you may assume that there are no limits on time complexity, message size, or storage space for either algorithm, and that you don't care about the message complexity of the pre-processing algorithm.

3. Give the best lower bound you can on the total message complexity of the pre-processing and search algorithms in the case above.

Solution

1. Run depth-first search to find the matching key and return the corresponding value back up the tree. Message complexity is $O(|E|) = O(n)$ (since each node has only $O(1)$ links).

2. Basic idea: give each node a copy of all key-value pairs, then searches take zero messages. To give each node a copy of all key-value pairs we could do convergecast followed by broadcast ($O(n)$ message complexity) or just flood each pair $O(n^2)$. Either is fine since we don't care about the message complexity of the pre-processing stage.

3. Suppose the total message complexity of both the pre-processing stage and the search protocol is less than $n - 1$. Then there is some node other than the initiator of the search that sends no messages at any time during the protocol. If this is the node with the matching key-value pair, we don't find it. It follows that any solution to the search problem. requires a total of $\Omega(n)$ messages in the pre-processing and search protocols.