CPSC 5520—Distributed Systems

Multicast Search Dances

Analysis

How would this be implemented in code?

1. These algorithms typically have a time-to-live timer on each query to prevent them from echoing around the group forever. What mechanism in our dance choreography represents the time-to-live timer?

   The query stacks. For the rumor spreading algorithm, TTL is the number of sticky notes in the original query stack. For the flooding algorithm, TTL is $\log_3(27) = 3$. This is because we had 27 note cards and split them in three each time—we stopped when only one card was left.

2. Both algorithms require the dancer to check their memory and maybe refuse the message. How would you implement this in code? How would you keep memory from growing without bound in your scheme?

   Keep a hash table or list of all the queries seen recently. We'd only need to keep an item in the list around for the TTL of that query. Once the TTL is over, there is no chance of seeing that query from a partner again.

3. What happens when the querier gets back more than one result? Do you have any ideas for preventing this? Do we care?

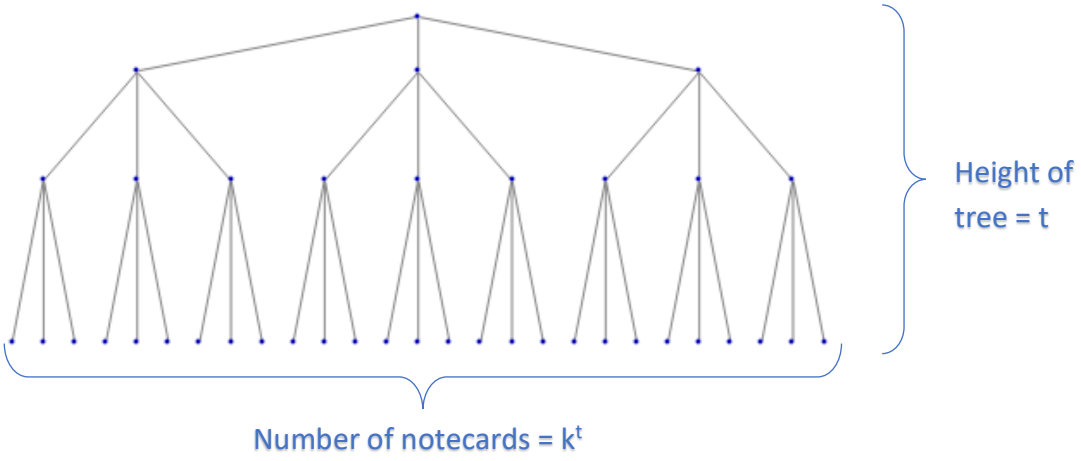
   Typically, we don't care. The second and subsequent copies can be ignored by the querier. We assume the data from each replica is identical (though for some systems it may be allowable to have some inconsistencies).

What is the cost and time of this algorithm where *n* is the number of nodes, *k* is the average number of known neighbors, *m* is the average number of replication sites for a typical object, *t* is the time-to-live, and we are measuring one query? For our dances, $n = 40, k = 3, m = 2$. Remember to think about worst and average cases if they are different.

4. Number of messages? Remember that in our dances, we started out with $27 = 3^3$ cards. Also remember that a refusal counts as a message.

   Number of messages for rumor spreading is simple. Worst-case is exhausting the TTL and not finding the data, in which case there are t messages. For average-case, we'd expect to find the data in n/m tries, so n/m messages on average. Refusals add a bit of complication but would likely be immaterial for large n.

   Number of messages for flooding is more complex. We can start with some simplifying assumptions to get a handle on it. First of all, note that our 27 notecards was $k^t = 3^3 = 27$, since at each splitting we multiply the number of nodes considering the query by k = 3 and we do this t = 3 times (that's why I started the queries with 27 notecards). To simplify, let's assume that the data is never found and that there are no refusals. Then the query promulgates like a k-ary tree (e.g, diagram for our dance with k=3 and t=3):



   Total number of edges in this tree is $k + k^2 + \cdots + k^t = \sum_{i=1}^{t} k^i \approx \frac{k}{k-1} k^t$
   (In our k=3, t=3 case above, the number of edges/messages was 39 which is pretty close to (3/2)*27. For the derivation of the summation result, see something like en.wikipedia.org/wiki/Geometric_series#Sum.)

   If we relax the simplifying assumptions, the math is much more complicated since we'd have some refusals (adding extra messages), some found data (removing all the tree nodes below them), and some failures to find exactly k peers to pursue the query, all of which will skew the tree.

5. Elapsed time? Report your answer in multiples of average message time.

   Elapsed time is just t in the flooding case since we assume there are some branches that will go to the end without finding the data.

   Elapsed time for rumor spreading is exactly the same analysis and results as for cost, since each time step is just one message. So t in the worst case and n/m on average.