CPSC 5520—Distributed Systems

Multicast Search Dances

Purpose

In many peer-to-peer and clustered systems, we need to find a node that can supply a given resource, like an object or the value for a key-value pair. We would typically expect the resource to be replicated on multiple nodes in the system, so finding any one of these replicas is sufficient. In these circumstances, we delve into the set of nodes with the key we are searching for and expect to get a direct response from one of the replicas saying they have what we are looking for (they may also actually respond with the resource in the same message if that is suitable). There are many techniques for finding a suitable replica, including DHT's. The techniques we will be dancing today will be peer-to-peer flooding and gossip-based searches.

Topology

This is a peer-to-peer overlay where every node knows a few neighbors. The querying process knows at least one active node.

Equipment

- Data elements are represented by playing cards. We will use two standard decks, so each one of the 52 cards is typically replicated at two nodes. The cards are dealt out randomly at the start of the dance.
- Search requests are written on query stacks—piles of notecards. The notecard at the top of the stack has the original querier's name and which playing card they are searching for, e.g., "Polly wants the 4♠" is written on the top notecard. When the query stacks are split, the dancer doing the split copies the top notecard's message to the top notecard of each of the split stacks without modification.

Protocol for Flooding Algorithm

When a dancer is dealt a query stack, she:

1. Checks her memory to see if she has already seen this query. If so, she refused the stack.
2. Then she checks if she has the queried playing card.
3. If she has it, then she establishes a connection to the original querier and delivers the requested playing card to him.
4. If she does not have it, she splits the stack into three and copies the query verbatim onto the top of the two new stacks. She then does her best to pass each stack to a neighbor.

Protocol for Rumor Spreading

When a dancer is dealt a query stack, she:

1. *similar to flooding algorithm*—checks her old notecards and refuses if this is old news
2. *same as flooding algorithm*—checks if she has the requested playing card
3. *same as flooding algorithm*—delivers it if she has it
4. If she does not have it and the query stack is only a single notecard. She does nothing more for this query.
5. If the query stack has multiple notecards, she takes off the top notecard and keeps it. She copies its message onto the top of the remaining query stack. She then does her best to pick one of her neighbors at random who doesn't refuse and passes the query stack to him.

Dances

1. Run the Flooding Algorithm. Pick one querier who starts a query stack 27 notecards and a query of their choice. He passes the query stack to one of his neighbors chosen at random.
2. Repeat experiment #1 a few more times.
3. Run the Rumor Spreading Algorithm. Proceed as in #1 and repeat a few times.
4. Run three *simultaneous* queries using the Rumor Spreading Algorithm.

Document your observations.

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?

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?

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

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, $n = 34, k = 3, m = 2$. 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.

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