CS 273: Kris Hildrum Notes

Scribe David Garmire

4/14/2003

Agenda

- Two object location systems (CAN and Tapestry/Pastry)
- Review of object location systems
- Finding nearest neighbor (Karger and Ruhl)

Two object location systems

CAN: Content Addressable Network

We want to find an object in a distributed network. Instead of going to a specific node, query the network for an object "location" and have the network take us to the right location. Take a space (in two dimensions for example), divide up into a grid and each node controls part of the grid. To find (15,15) from (10,10), go up in x and up in y. (Refer to figure 1.)

Q:What do you mean when you say the grid neighbors have nothing to do with where you are in the network?

A:You give each node a range (not necessarily assigned based on physical location). Two nodes far apart in the network could get adjacent nodes in the grid.

In order for this to work, go up in x and go up in y, what does everyone need to store? You need to store the neighbors (in the grid). If you have d dimensions, how many neighbors do you need to store? You need to store about $2d$ neighbors.

Are routes going to be short? In two dimensions, your routes could be size $\sqrt{n}$ where n is the number of nodes. In general, it takes about $2d$ guys and $dn^{\frac{1}{d}}$, if you use $d = \log n$, there are not very many hops required.

$$\begin{aligned} dn^{\frac{1}{d}} &= n^{\frac{1}{\log n}} \log n \\ &= e^{\frac{1}{\log n} \log n} \log n \\ &= e \log n \end{aligned}$$

How do we handle a new node? We can split the grid in half (each node will control a smaller square). As you get more nodes, the ranges become smaller and smaller. You do not want one node to control a very small range and another node to control a very large range (very poor load balancing).

Q: You are essentially doing quad trees?

A: Yes. This is the idea in two dimensions (and generalized to d dimensions).

Tapestry/Pastry

(Refer to the tree drawn in Figure 2.)

Give nodes random IDs in base B, for this example assume $B = 5$. For every prefix α of an ID, store the closest node with ID αj, where $j \in [0, b]$. What is the smallest prefix of 1234? The empty set. Let's build a table of the possible prefixes.

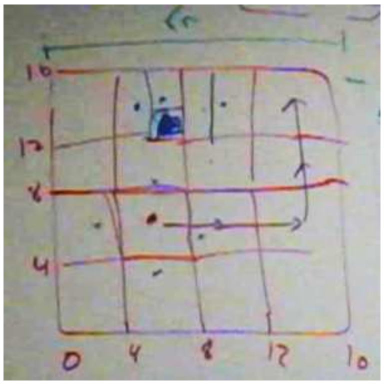

Figure 1: Illustration of Grid in CAN

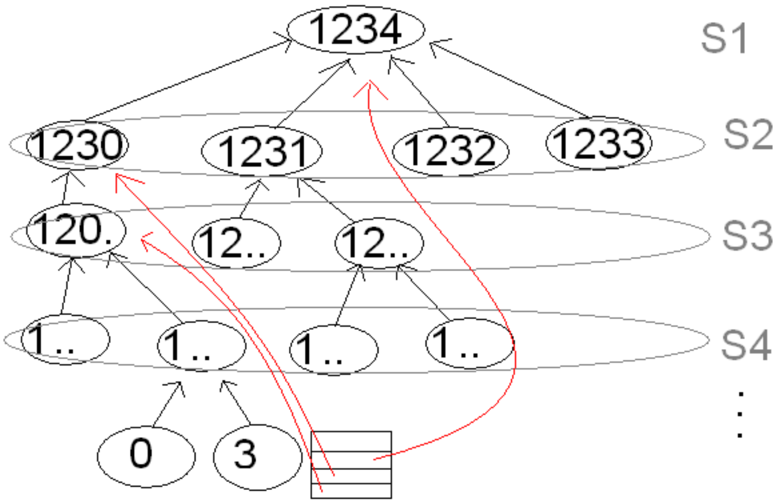

Figure 2: Illustration of Tapestry Tree

We are not going to show how to deal with getting to exact node.

If you want a particular ID, how do you find it? The problem here is you have nodes and you have assigned them random IDs. I am going to tell you how to route them. Suppose you are at 1234 and want 4321? Suppose we go to 4112. What is this 4112 going to do? (It will also have a table and has 43?? in it.) We do not know what it will have in its table – all we care about is some node beginning with 43.

Q: Do you need a lot of tables? Do you need a table for every possible input?

A: Every node has its own table. The cool thing about this method is this is the only table you need at a given node.

In the next node, you would match the 432. If you are lucky, you could match more nodes. (You could think of this as sending a message to yourself.)

Q: When do you know that the second column you have starts with 12?

A: Your table depends on your ID. Everyone only needs to keep one table.

For 1234, your table might look like:

	120.	10..	0...
	121.	11..	1...
1232	122.	12..	2...
	123.	13..	3...
		14..	4...

When we have a choice, pick the closest possible prefix.

Q: Why are some of the nodes on the 12 level are pointing to the 1234?

Q: Are those nodes chosen arbitrarily?

A: Where are these arrows coming from? I am redrawing these neighbor tables from a different point of view. Pick the closest node with given prefix. (So it is determined.)

Let me point something out. Call each of these levels S_1, S_2, ... S_k. This scheme resembles subsampling where we get the closest in each previous subset. The difference here is that instead of storing the closest in all previous samples, we're storing the closest in just the first previous subsample. (We are assuming that we can infer the closest on the higher subsamples from traversing up the tree.)

How much you have to pay is constant. The closer you start to an object in the tree, the more quickly you find it (in nice metric spaces like a grid).

Q: Did you just arbitrarily pick the roots to be 1234?

A: Yes and no, I did pick the roots 1234 in this picture, but I could draw this picture for any object. For any object, this tree exists. Lets look at the nodes surrounding this node.

Q: How do you know how much space is controlled by a node?

A: You could use consistent hashing, the computer could have $\log n$ IDs, or any number of things to balance. These systems solve it in different ways. (Also existence problem.) Not a particularly fascinating problem. You have to think about it and be careful, but not particularly clever. (Discussed in papers.)

Review of object location systems

Name	Idea	Publish	Stretch	Hops	Storage	Balanced	Dynamic
Awerbuch-Peleg [1]	Sparse Covers	$\log^2(n)\log(\Delta)$	$\log n$	$\log\delta$	$\log^2 n$	with [4]	No
Thorup-Zwick [7]	Subsampling	$\log n$	$\log^2 n$	$\log n$	$\log n$	No	No
Chord [6]	Consistent Hashing	$\log n$	-	$\log n$	$\log n$	Yes	Yes
CAN [5]	Grid	$dn^{\frac{1}{d}}$	-	$dn^{\frac{1}{d}}$	d	Yes	Yes
PRR Tapestry [3]	Prefix Tree	$\log_b(n)$	Maybe O(1)	$\log n$	$blog_b(n)$	Yes	Yes
Central Hash Table	411	Maybe O(1)	-	1	1	No	Yes, hard
Everyone's Equal	Phone Book	O(n)	1	1	O(n)	Yes, big	Yes

The schemes used in the internet do not do anything about the delay. The schemes that are dynamic are the ones that do not have any notion of delay. The schemes that have a notion of delay, are not dynamic.

For Tapestry, the stretch of O(1) was proven to work in networks of a special type (E.g., grids and see next section).

Finding nearest neighbor

Instead of doing object location (where we can miss a node and go back to it), we are actually trying to find the closest node.

It, in general, is a hard problem but it has some feasible approaches if special constraints are imposed [2]. We will deal with only the growth restricted metric. Let $B(x, r)$ denote a ball around x of radius r whose magnitude, $|B(x, r)|$, represents the number of nodes in the ball. For all x and for all r, $|B(x, r)| \leq d|B(x, r)|$. A metric space is a means to measure the distance between a set of points. If you have a graph, that gives you a metric space. (Distance between a node and itself is 0.) You could have finite metric spaces, as well. A grid is a growth restricted metric. Not all graphs are growth restricted metrics: for example, a four clique is a graph that is not a growth restricted metric.

We're at p, want closest point to q. Let $r = d(p, q)$. Suppose we could find some point p' such that $d(p', q) \leq \frac{r}{2}$. Then we could repeat and we eventually find the closest. Why do we know we could even possibly find such a point? Why might this idea not be reasonable?

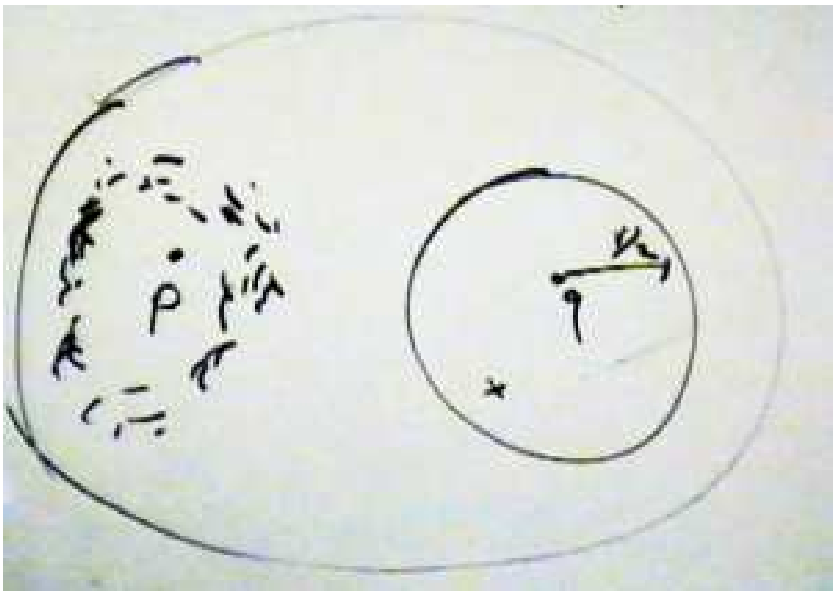

Figure 3: Illustration of Non-growth Restricted Problem

What if there is nothing that is closer? What if we placed a large number of points around p, then we would have to sample possibly infinitely long before reaching q. (Refer to Figure 3.) Fortunately, we are in a growth restricted metric, so this bad case will not occur.

What would a ball of radius r around p do? It would just include the tip of q. We could use $\frac{3}{2}$, but no reason to choose this fraction, just use 2 for convenience. (Refer to Figure 4.)

Let us sample uniformly at random around $B(p, 2r)$ to find a closer point that is a point in $B(q, \frac{r}{2})$.

We are not finding a good path from p to q, we are iteratively shortening the distance from p to q by varying p. We want to find the closest point to q. (Maybe q wants to enter metric space.)

Consider $B(q, 4r)$. You want to find someone in q's ball. Note that this ball completely contains $B(p, 2r)$. Suppose x nodes in $B(q, \frac{r}{2})$. Then there are at most c^3x nodes in $B(q, 4r)$ and in $B(p, 2r)$. Then there are $\frac{1}{c^3}$ nodes in $B(p, 2r)$ that are also in $B(q, \frac{r}{2})$.

If we are using $3c^3$ samples, then with probability $\frac{9}{10}$ we find someone in the blue ball, $B(q, \frac{r}{2})$. The proof follows from the fact that the probability you do not find anyone in the blue ball is $(1 - \frac{1}{c^3})$, so the probability all fail is $(1 - \frac{1}{c^3})^{3c^3}$ which is less than or equal to $e^{-3} \approx \frac{9}{10}$.

References

[1] Awerbuch, B., and Peleg, D. Concurrent online tracking of mobile users. In *Proc. of SIGCOMM* (Sept. 1991), pp. 221–233.

[2] Karger, D., and Ruhl, M. Find nearest neighbors in growth-restricted metrics. In *Proc. of the 34th Annual ACM Symp. on Theory of Comp.* (May 2002), pp. 741–750.

[3] Plaxton, C. G., Rajaraman, R., and Richa, A. W. Accessing nearby copies of replicated objects in a distributed environment. In *Proc. of the 9th Annual Symp. on Parallel Algorithms and Architectures* (June 1997), pp. 311–320.

[4] Rajaraman, R., Richa, A. W., Vöcking, B., and Vuppuluri, G. A data tracking scheme for general networks. In *ACM Symposium on Parallel Algorithms and Architectures* (July 2001), pp. 247–254.

[5] Ratnasamy, S., Francis, P., Handley, M., Karp, R., and Schenker, S. A scalable content-addressable network. In *Proc. of SIGCOMM* (Aug. 2001), pp. 161–172.

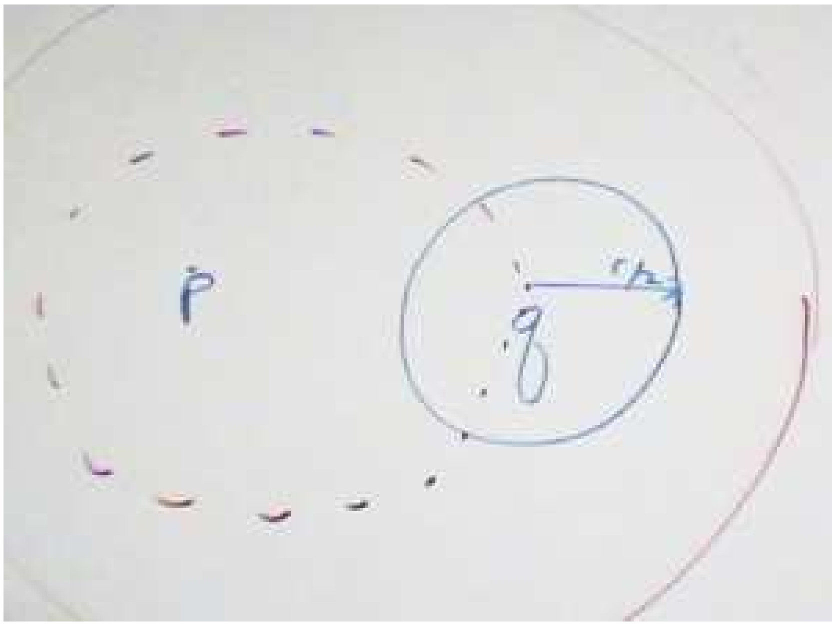

Figure 4: Sample Large Solid Circle and Want Points Inside Small Blue Circle

[6] Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., and Balakrishnan, H. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. of SIGCOMM* (Aug. 2001), pp. 149–160.

[7] Thorup, M., and Zwick, U. Approximate distance oracles. In *Proc. of the 33th Annual ACM Symp. on Theory of Comp.* (July 2001), pp. 183–192.

[8] Hildrum, K., Kubiatowicz, J. D., Rao, S., Zhao, B. Y. Distributed Object Location in a Dynamic Network. In *Proc. of the Fourteenth ACM Symposium on Parallel Algorithms and Architectures.* (Aug. 2002), pp. 41–52.

[9] Zhao, B. Y., Kubiatowicz, J. D., and Joseph, A. D. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Tech. Rep. UCB/CSD-01-1141, UC Berkeley, Computer Science Division, Apr. 2001.