CptS 464/564
Lecture 28

7th December 2004

References for today's lecture

A. Rowstron and P. Druschel, *Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems,* Proc. of the 18th IFIP/ACM International Conference on Distributed Systems and Platforms (Middleware 2001), Heidelberg, Germany, Nov. 2001.

S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker, *A Scalable Content-Addressable Network,* ACM SIGCOMM '01, San Diego, August 2001.

Grad students should obtain and read these papers. As always, read not only for the content but also try to discern qualities of the writing such as the organization of the paper, structure of citations, etc.

Peer-to-peer systems

The late 1990's saw the rise of a number of internet file-sharing systems such as Gnutella, Napster and FreeNet. Such *peer-to-peer* systems are characterized by nodes with identical capabilities and responsibilities along with symmetric communication.

- different from client-server in that nodes may take active and passive roles in the same protocol

The simulated system in Hwk 4 is a simple example of a peer-to-peer system.

A fundamental problem in peer-to-peer systems is how can they be organized so that only a small amount of management effort is required even as the system grows to millions of nodes. Organizing a force of a million agents is a formidable challenge: consider that there are few if any private companies with even a million employees, few governments with a million employees, few armies with a million soldiers; furthermore, when these organizations do exist they have typically grown over many years. Yet worldwide, societies accomodate a growth of 79 million new people per year. Decentralized control, self-organization, and adaptability seem to contribute a scalability not available in more highly-organized systems.

What techniques can provide these properties for peer-to-peer systems?

Example

We will use a *distributed hash table* as our example. Recall that a hash table stores *(key, value) pairs* with the purpose of providing fast access to the value given the key. (Fast meaning faster than looking through all the existing pairs). How can a set of (key, value) pairs be stored on a collection of hosts so that any pair can be quickly found?

Non hash-table: we've seen one approach when we looked at name services: a hierarchical arrangement of names and a matching hierarchy of servers such as DNS and GNS. Such systems suffer from high management overhead to organize the name space: hierarchy must be agreed on, ownership of parts of the hierarchy settled, continuing maintenance, etc.)

But can we build distributed hash table and avoid the hierarchy and management? Yes!

How? Instead of organizing our hosts and keys, we will dis-organize them using randomization. Assign to each host a random host id from a large name space, e.g. numbers in $0..2^{128}-1$. The random ID can be obtained by flipping a coin 128 times or cryptographically hashing the node's IP address, or ... It doesn't really matter as long as the ID is random.

For any conceivable number of hosts, this address space will be sparsely populated. Further, node IDs will be uncorrelated with any other node attribute such as geographic location, IP address, ownership, etc. (dis-organized in the extreme!)

Now, if we have a hash function that maps our keys into $0..2^{128}-1$ as well, we store each (key, value) pair at node *n* such that $|hash(key) - ID(n)|$ is minimized. Again, the placement of the data is uncorrelated with any other attribute of the nodes. That might be bad: for example, if it's not correlated with the node's IP address how can the correct node be found in order to lookup a key?

Overlay networks

To answer that question we need to introduce the notion of an *overlay network*. An overlay network is a graph whose nodes are the same as those of an underlying network (such as the internet) but with a different set of edges. Two nodes connected by an edge in the overlay network may be (and likely are) far separated in the underlying network. Nevertheless, messages can be sent between the nodes, as if they were adjacent, by using the underlying network. The edges of the overlay network are virtual – no wires correspond to them, so they are very inexpensive to have (but expensive to use). Therefore, we get to design the overlay network with precisely the edges we want in order to serve the purpose at hand. (And, in keeping with our earlier remarks about scalability, it would be very useful if we didn't have to actively design the network but instead could use local decision making yet nevertheless achieve desirable global properties).

Returning to the DHT problem: given a key, what must an overlay network do to route a message to the node with the closest ID to hash(key)?

Suppose each node has several edges on which the message might be sent and that every node is reachable (in the overlay network) from every node. Then if each node will simply send a message to its neighbor that is closest to the destination address, the message eventually reaches the node that is closest to the destination.

So how should we choose the neighbors for each node. We'll look at two possibilities – Pastry and CAN. There are others.

Pastry

Pastry chops the node IDs into a sequence of b-bit "digits" and organizes its routing table as $log_{2^b}N$ rows of $2^b - 1$ entries each. Entries in the n^{th}row of the table share the first $n-1$ digits with the current host ID and differ in the n^{th}. Some rows may be empty. The node joining protocol populates the routing table from those

of nodes on the path the nodes nearest neighbor (in host ID space). Nodes also have a so-called "neighbor table" in which they record the IDs (and IP addresses) of their L/2 nearest upper neighbors and L/2 nearest lower neighbors (again in host ID space). The neighbor set is used to make the final steps to the destination and also to make progress when a needed routing table slot is empty.

It is not too hard to show that the expected path length *in the overlay network* (that is, along overlay edges) is $O(log_{2^b}N)$ steps. However, remember that the overlay edges could each traverse many links of the underlying IP network. The primary administrative task for Pastry is to help a joining node find an existing Pastry node, preferably nearby in IP space. Once that is done, the algorithms can manage the network.

The creators of Pastry came up with a clever way to make use of IP routing distance metrics in constructing the overlay routing tables. Basically, what happens is that amongst nodes that have the right node ID prefix, the algorithm prefers ones that are nearby to insert in the routing table.

By incorporating the distance metrics, experimental results show that Pastry messages travel, on average, somewhat less than twice as far as messages sent directly at the IP layer from the source to the destination.

CAN

CAN stands for "Content Addressable Network". As in Pastry, nodes choose random addresses, and keys are hashed to random values. IDs are interpreted as coordinates in a d-dimensional space. Each node is thought of as "owning" some subspace of the d-dimensional space.

When a new node joins, its ID falls in the subspace owned by one of the existing nodes. The subspace is split in half in one dimension, with the new node taking ownerhship of one half, the old node keeping the other half. Existing (key,value) pairs are partitioned between the two nodes depending on which half of the subspace they lie in.

The CAN routing table at each node contains the coordinate spaces and IP addresses of each of its neighbors in ID-space. Two nodes are said to be neighbors if their subspaces overlap in $d-1$dimensions and abut (are adjacent) in the final dimension.

Example: see figure 2 of the paper.

Using this routing scheme a d dimensional CAN network requires $O(d)$ space at each node for routing tables (contrast $O(log$ n) for Pastry) and expected $O(n^{1/d})$ routing hops in the overlay network between source and destination. With $d = 4$, and clever choice of which neighbor entries to use for routing, experiments with simulated CAN networks also show that messages travel less than twice as far as in the underlying IP layer.

Summary

Overlay networks are a hot research area right now. Their appeal is reduced management cost, simple, localized routing algorithms. These properties are important for networks supporting services such as distributed hash tables and file replication. What we pay for this convenience is additional hops (at the IP layer) traversed by each message, which means also increased total traffic at the IP layer.