CS273 Foundations of Parallel and Distributed Systems **Spring 2003**

Lecture 18: Apr 2, 2003

Lecturer: Satish Rao *Scribe: Kamalika Chaudhuri*

18.1 Outline

In today's lecture, we shall talk about tree covers of a graph, and how they lead to good routing schemes in a network.

18.2 Motivation: Routing Systems

Suppose one has been given a huge network spanning thousands of nodes and a large geographical region, where any node can contact any other node and nodes may move as well. The problem is to design a routing scheme for such a network, that neither involves too much space nor too much routing overhead. Let us first look into the requirements of such a routing scheme. Such a routing system should support two operations - (a) Insert, where a node tells the system its location and (b) Lookup, where a node queries the location of another node. The cost of each operation is measured by the time it takes to complete the operation, as well as the number of messeges involved. Another measure of the cost of a scheme is also the total space needed by the routing tables (at all nodes). Several simple routing schemes naturally come to mind.

Single Directory

A single global directory is maintained, which contains location information about all the nodes in the network. Under such a routing scheme, insertion takes $O(D)$ time, where D is the diameter of the network. Only one messege needs to be sent for insertion to the global directory. Lookup also takes $O(D)$ time in the worst case, and involves only one messege. The total space taken up by this routing scheme is $O(n)$, since only the global directory contains any routing information.

Broadcast

Under this scheme, a node informs all other nodes of its location, and every node knows the location of every other node. Insertion still takes $O(D)$ time, but now we need to send $O(n)$ messeges, where n is the number of nodes in the network. The total space needed by this scheme is $O(n^2)$.

18.3 Routing on the Straight Line: Covering the Straight Line with Trees

The single directory method suffers from the drawback that, even if a node, say A, wants to contact another node B which is very close to A, it still has to go all the way to the directory to find out its location. This ideally should not be the case.

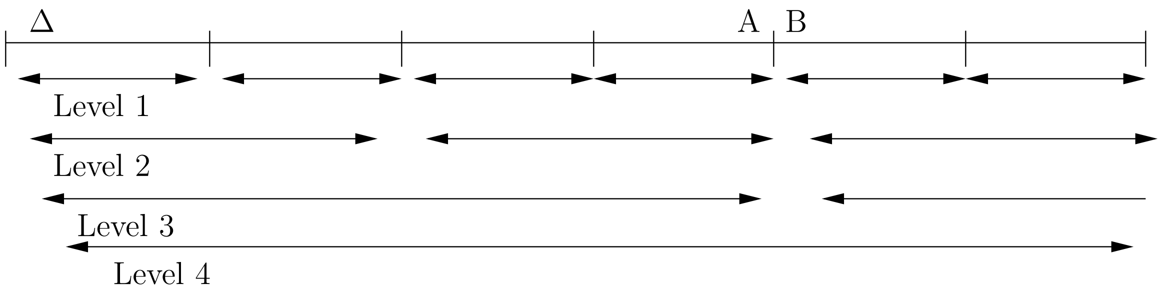

Figure 18.1: A Hierarchical Routing Scheme for nodes on a straight line. The nodes A and B are the problem nodes, which need to go all the way to the top level directory to lookup each other.

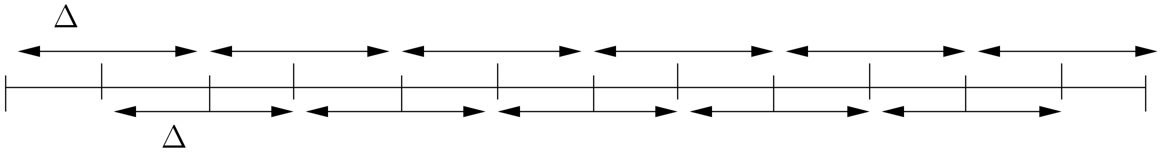

Figure 18.2: A Better Hierarchical Routing Scheme for nodes on a straight line

To remedy this, we will design a hierarchical routing scheme. For the purposes of the current scheme, we will assume that all the nodes lie on a straight line. Directories are maintained at several levels. We divide the straight line into segments of length Δ. Location information about all the nodes in a segment is stored in a level 1 directory, located say at the center of the segment. A level 2 directory duplicates information contained in two adjacent level 1 directories, a level 3 directory duplicates the information contained in two level 2 directories and so on. Finally there is a global directory, which contains all the information about all the nodes. The hierarchical directory scheme is shown in Figure(20.1).

Let us look at the cost of this scheme. At each level, the total space required by all the directories is $O(n)$ and there are at most $\log n$ levels - the total space requirement is therefore $O(n \log n)$. Insertion of an entry still takes $O(D)$ time, since we need to go all the way to the top level directory; however, there are now only $O(\log D)$ messeges to be sent. For lookup, our aim is to reduce the time to lookup nodes which are close to each other. Consider two nodes A and B, both situated within a distance Δ of each other.If A and B lie within the same segment, it will take A only constant time and one messege to look up B, which is what we want. However, if A and B are situated as shown in Figure(20.1), A still has to go all the way to the top level directory to lookup B.

This can be fixed by maintaining not one but two sets of directories. The segments for the two sets of directories overlap as shown in Figure(20.2); and information about all the nodes in a segment is stored at the center of the segment. Note that in this scheme, the $\Delta/2$ neighborhood of any node lies within a single segment; also note that information for each node now occurs in two segments at each level.

18.4 Extension to General Graphs

This idea leads to the following hierarchical routing scheme for general graphs. First we want to find a tree cover for a general graph in the following manner. Given a value of Δ, we want to find a set of trees of diameter at most $\gamma\Delta$, such that for every node, its Δ neighborhood lies in one tree, and no node occurs in more than α trees. Here, α and γ are parameters. Once we can do this for some values of α and γ, we can build up a hierarchical routing scheme as follows. For $\Delta = 1, 2, 4, 8, \ldots$, we construct an α, γ tree cover of

Mark all nodes in G_Δ as uncovered and unremoved
do
  do
    Pick an uncovered, unremoved node $v \in V_\Delta$
    Cover(v)
  while there is an unremoved node $v \in V_\Delta$
  Put back the removed uncovered nodes into the graph
while there is an uncovered node $v \in V_\Delta$

Figure 18.3: Main Algorithm for finding a Tree Cover of a general graph

Cover(v) $S \leftarrow v$
If $|N(S)| \geq |S|$
  $S \leftarrow S \cup N(S)$
Else
  Remove $S \cup N(S)$
  Output a spanning tree of the vertices in the original graph present in S
  Mark all the nodes in S as covered

Figure 18.4: Procedure Cover

the underlying graph; then for each tree in the cover, we put the information about all the nodes in the tree at the root of the tree.

Insertion in this scheme would take $O(D)$ time, (since information is to be entered into the top-level tree) and involves at most $O(\log D)$ messages. Lookup takes time $O(\gamma\delta)$, where δ is the distance between the current node and the node being looked up; and involves $O(\alpha \log \delta)$ messages. The total space requirement of the scheme is $O(\alpha n \log n)$.

Now we shall look at an algorithm which finds such a decomposition in a general graph for parameters $\gamma = \log n$ and $\alpha = \log n$, and for any Δ. Let us define some notation first. Given a graph $G = (V, E)$, we will work with the graph $G_\Delta = (V_\Delta, E_\Delta)$. The vertices of G_Δ correspond to Δ-neighborhoods around the vertices of G. There is an edge between two such vertices u_Δ and v_Δ if the Δ-neighborhoods of u and v in G share a vertex. Define $N(v)$ as the set of neighbors of a vertex v; also define the neighborhood of a set of vertices as the union of their neighborhoods.

The algorithm is as follows.

Note that in the procedure Cover(v), $N(S)$ is used to denote the uncovered and unremoved neighborhood of S. At the end of each round of calls to the procedure Cover(), we have the following facts.

1. Each round of calls to Cover() covers at least $\frac{1}{2}$ of the total number of uncovered balls at the beginning of the round This follows easily from the stopping condition $|N(S)| < |S|$, and the fact that the uncovered nodes at the end of each round is the union of the $N(S)$'s.

2. No tree has diameter more than $\Delta \log n$ Since we at least double the number of balls in S at each round, there can be at most $\log n$ doublings; hence S can have diameter at most Δ and therefore the spanning tree made out of S can have diameter at most $\Delta \log n$

3. No node appears in more than one tree in a given round This happens because when a tree is made out of a set S, the nodes in S are removed and do not appear in any subsequent tree in the same round.

From fact (1), it is also easy to see that there are at most $O(\log n)$ rounds. Therefore, each node in the graph appears in at most $O(\log n)$ trees. Also the diameter of each tree is $O(\Delta \log n)$ - so the parameters of this decomposition are $\alpha = \log n$ and $\gamma = \log n$.