Randomized Routing

Scribe : Aditya Desai
Instructor : Anshumali Shrivastava

In this lecture we study the problem of routing messages in a cluster of nodes being used for distributed computing. Distributed computing has become pervasive in industry due to increase amounts of data and complexity of models. The idea is to distribute any large computation across different nodes (or machines) and reconcile the output to effectively solve the big computation. Such a protocol of computation needs communication between the nodes involved. Among different patterns of communications, all-to-all communication is one of the pattern. As the name suggests, in all-to-all communication, each node wants to send some piece of data (called message) to some other node.This pattern adds most communication to the network running into risk of congestion which leads to increase in running time of the entire computation as a whole. In this lecture we focus on all-to-all communication pattern and study routing algorithms to reduce delay caused by congestion.

Grid/Hypercube Architecture In very large clusters, every node cannot be connected to every other node. In such a case, nodes are connected to few other nodes and when a message has to be passed from one node to other, it has to determine a path to traverse. We consider a specific arrangement of nodes called grid model. Consider that we have $N = 2^n$ nodes in our graph. We identify each of the nodes with binary strings. We would need $log_2(N) = n$ bits to identify N different nodes. In grid model, the two nodes are connected only if they have hamming distance 1; i.e. their binary string ids have only one bit mismatch. So each node hash a degree equal to number of bits in its representation , i.e. n. In figure 1 a grid model for N=8 is shown. [1]

Routing Problem In this problem, we assume that each link can carry only one message at a time. Hence whenever two or more messages arrive at a link, a queue is formed which is processed in one by one fashion. This adds delay to the overall all-to-all communication step. The purpose of our algorithm

[1]This figure is taken from https://people.eecs.berkeley.edu/ jfc/cs174/lecs/lec11/lec11.pdf

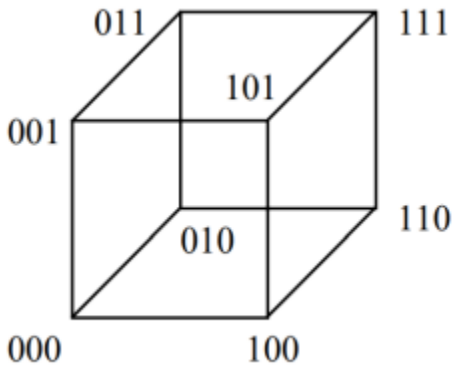

Figure 1: A grid with 8 nodes. It requires 3 bits for identification of nodes by binary strings. Also, we connect the nodes

Current	Destination	Next
11011	10101	10011
10011	10101	10111
10111	10101	10101
10101	10101	END

Figure 2: Sample run of the bit fixing algorithm

is to minimize the delay. In a way this is a load balancing problem : we want to make sure no link receives extreme load. However there is added difficulty in this setting. Given that we are implementing the algorithm over network, we have a severe restriction on memory and computation resources. To be practical, we assume that we want to have an memory less algorithm. Specifically, whenever a packet arrives at a node, it only carries the destination address and nothing else. This requirement rules out a lot of optimal algorithms. However this gives a perfect setup for randomized algorithms as we have seen in load balancing hashing algorithms.

Bit Fixing Algorithm

One-One message passing Before delving into the actual problem, its always advisable to look at simpler version of the problem to understand its properties. Lets look at the problem in which we want to pass a message from *start* to *dest* node. *start* and *dest* are bit string identifiers of the nodes respectively. We first present the algorithm and then discuss the properties of this algorithm.

Algorithm 1: Bit Fixing Algorithm

```
Result: Path of message from start to dest
current ← start
while current ≠ dest do
    next ← current
    lbit ← lowestMismatch(current, dest)
    next[lbit] = dest[lbit]
    send(current, next)
    current ← next
end
```

A sample run of the algorithm is shown in the figure 2, where the start=11011 and destination=10101. At each step the first mismatch in bits is underlined.

- ***Bit fixing Algorithm is memory less***. In order to compute the link on which to send to message, at each step, the algorithm only requires destination address and its own current address (which it knows). So it satisfies the requirement of memory-less we defined above.
- ***Bit fixing Algorithm has optimal substructure property.*** If you consider the any sub path (say $node_i$ to $node_j$) of the path bit fixing algorithm outputs for (start, dest) is exactly the path it would output for $start = node_i$ and $dest = node_j$. So in the bit fixing algorithm, there is exactly one path between any two nodes, irrespective of what the start, dest nodes are.
- ***Identity of path nodes***. While going from start $(S_1, S_2, \ldots, S_n)$ to dest $(D_1, D_2, \ldots D_n)$, every intermediate node has the bit string representation $(D_1, D_2, \ldots D_i, S_{i+1}, \ldots S_n)$ And with each progressing step the prefix of intermediate node matching with destination only grows longer. This also establishes that there are no cycles in the path.

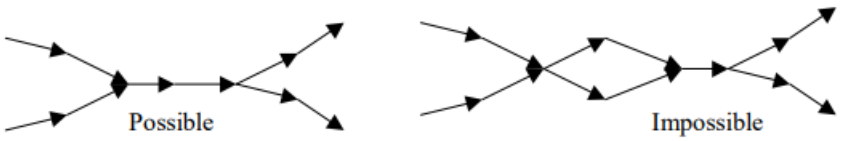

Figure 3: Property of Bit fixing algorithm for two paths for pairs (s1,d1) and (s2, d2)

- ***Path interaction***. The above properties also add restrictions to the interactions of two paths say (s1,d1) and (s2,d2) can have. Specifically, the paths can join each other at at most 1 node and continue on that path till they diverge. They cannot diverge and join each other again. We will use this property in our analysis for randomized routing. This is clearly shown in figure 3 [2]

Permutation Routing

Now lets look at all-to-all routing. This scheme is also called as permutation routing. For the sequence of start nodes $starts = [1 \dots N]$, we can specify the destination nodes as a permutation of the start sequence $dests = Perm([1 \dots N])$. We have to pass the message from $starts[i]$ to $dests[i]$ for each i. The following theorem gives an idea of how well a deterministic algorithm can perform.

Theorem 1 *Any (memory) oblivious deterministic algorithm for permutation routing with N machines and n ($\approx logN$) outward links require $\Omega(\sqrt{\frac{N}{n}})$ steps*

This theorem basically requires us to look for solutions beyond the deterministic realm of algorithms. We will see that the results with the randomized routing will help us break this bound at least probabilistically.

Randomized Routing

We will use bitfixing algorithm as a basic routing algorithm. If we were to run bitfixing algorithm on permutation routing problem , how would we expect the algorithm to fare? It is very obvious that an adversarial selection of permutation of starts can cause a lot of delays in routing. However, we would expect the delays to be bounded if the permutation was random. Before moving to the final algorithm, lets evaluate if our intuition is correct.

Theorem 2 *If dests is random permutation of starts, then with probability $(1 - (\frac{1}{2})^{1.5n})$ message from starts[i] reaches the dests[i], for all i, in no more than 4n steps*

Proof

$$Time[Packet[i]] \leq n + delay_i \tag{1}$$

where delay is caused by different messages intersecting the path of this message from starts[i] to dests[i]. We can bound the delay as

$$delay_i \leq \sum_j (Intersections(i, j)) \tag{2}$$

As defined in the problem above, the delay occurs only when two or more packets want to use the same link. Hence the paths of i and j must intersect for j to add to the delay for i. One important thing to note is that every packet can at most add a delay of 1 unit to the other packet. This is because a packet j can

[2]The figure is taken from https://people.eecs.berkeley.edu/ jfc/cs174/lecs/lec11/lec11.pdf

only *join* the path of i at one single point. This follows from the optimal substructure property of bit fixing algorithm. Also, while they are sharing the path, a delay is added only at the start of the shared path. So we can write the intersection as an indicator.

$$delay_i \leq \sum_j \mathbb{I}(Intersection(i,j)) \tag{3}$$

Let us look at the expected value of this indicator variable under the assumption of the random permutation.

$$\sum_j \mathbb{I}(Intersection(i,j)) \leq \sum_{e \in Path(i)} T(e) \tag{4}$$

T(e) is the total number of nodes sharing the edge e with node i. Note that R.H.S is an overestimate as each Indicator is replaced by the number of edges shared between the two paths. Taking Expectation of $delay_i$, we have

$$E(delay_i) \leq \sum_{e \in Path(i)} E(T(e)) \tag{5}$$

As, number of edges is bound by n,

$$E(delay_i) \leq nE(T(e)) \tag{6}$$

in order to find the E(T(e)), consider the following equation counting T(e)

$$T(e) = \sum_{j \in [1,N]} (\mathbb{I}(e \in Path(j)) \tag{7}$$

$$\begin{aligned} \sum_{e \in allEdges} T(e) &= \sum_{e \in allEdges} \sum_{j \in [1,N]} (\mathbb{I}(e \in Path(j)) \\ &= \sum_{j \in [1,N]} \sum_{e \in allEdges} (\mathbb{I}(e \in Path(j)) \\ &= \sum_{j \in [1,N]} (length(Path(j))) \end{aligned} \tag{8}$$

Taking expectation

$$\begin{aligned} &E(\sum_{e \in allEdges} T(e)) = \sum_{j \in [1,N]} E(length(Path(j))) \\ &NnE(T(e)) = N\frac{n}{2} \\ &E(T(e)) = \frac{1}{2} \end{aligned} \tag{9}$$

Hence expectation of the delay is

$$E(delay_i) \leq \frac{n}{2} \tag{10}$$

As $delay_i$ is a sum of independent bernoulli variables, we can use chernoff bounds to obtain the tail bound.

$$P(delay_i \geq (1+\delta)\mu)) \leq e^{-(\frac{\delta^2 \mu}{2+\delta})} \tag{11}$$

where $\mu \leq \frac{n}{2}$ Now we want to choose δ in a way that rest of the analysis follows, specifically, we want the probability on the right to be small enough to give a good tail bound after we apply the union bound.

Lets aim for now to keep the probability on right bounded by $\frac{1}{N^{2.5}}$

$$\begin{aligned} e^{-(\frac{\delta^2\mu}{2+\delta})} &\leq \frac{1}{N^{2.5}} \\ -log_2(e)(\frac{\delta^2\mu}{2+\delta}) &\leq -2.5log_2(N) \\ log_2(e)(\frac{\delta^2\mu}{2+\delta}) &\geq 2.5n \geq 2.5 * 2\mu \\ log_2(e)(\frac{\delta^2}{2+\delta}) &\geq 5 \end{aligned} \tag{12}$$

By substituting $\delta = 5$ we can see that the equation above is satisfied. Hence we have,

$$P(delay_i \geq 6\mu)) \leq \frac{1}{N^{2.5}} \tag{13}$$

$$P(delay_i \geq 3n)) \leq \frac{1}{N^{2.5}} \tag{14}$$

Applying union bound over all the nodes, we get

$$P(\exists i \quad delay_i \geq 3n)) \leq \frac{1}{N^{1.5}} \tag{15}$$

Hence the total time according to equation 1, is bounded by

$$P(TotalTime \geq 4n)) \leq \frac{1}{N^{1.5}} \tag{16}$$

Hence with probability $1 - \frac{1}{N^{1.5}} (= 1 - (\frac{1}{2})^{1.5n})$, the total time taken is no more than 4n
That completes our proof.

Solution to general *starts, dests*

So from the above theorem, it is clear that our intuition about random destinations is actually correct. We can use this to bound the time of any permutation routing problem. This can be done by adding a random intermediate destination between *starts* and *dests* locations.

$$starts \longrightarrow dests$$

The above problem can be converted to the following and we use bitfixing algorithm for both the sections of the problem.

$$stats \longrightarrow random \longrightarrow dests$$

Using the theorem 2, we can say that with high probability the total time taken by the routing algorithm is bounded by 8n.