Basic Problems and Protocols

Parallel and Distributed Systems

Jukka Viinamäki

February 4, 2007

Basic Problems and Protocols

Properties of basic protocols

- Basic: Commonly required for the functioning of the system
- Primitive: Often a module of more complex protocols
- Assume standard restrictions $\mathbf{R} = \{\text{BL}, \text{CN}, \text{TR}\}$

Broadcast: The Problem

- Consider a distributed computing system where only one entity, x, knows some important information; this entity would like to share this information with all the other entities in the system. This problem is called broadcasting (**Bcast**)
- Inherent in the problem is a new restriction: Unique Initiator with the information, UI+

Flooding revisited

Status Values: $S = \{\texttt{INITIATOR, IDLE, DONE}\}$
$S_{\texttt{INIT}} = \{\texttt{INITIATOR, IDLE}\}$
$S_{\texttt{TERM}} = \{\texttt{DONE}\}$
Restrictions: `BL, TR, CN, UI`

`INITIATOR`
Spontaneously
begin
send(M) **to** N(x);
become `DONE`;
end

`IDLE`
Receiving(M)
begin
send(M) **to** N(x) - {**sender**};
become `DONE`;
end

Problem Complexity

- `Flooding` uses $O(m)$ messages and has worst case ideal time of $O(n)$
- But what is the complexity of the broadcasting problem?
- Establish a lower bound f, such that the cost of each solution is at least f
- The problem complexity is irrespective of the solution algorithm

Worst case ideal time complexity

- Measuring ideal time complexity, assume Unitary Transmission Delays and Synchronized Clocks
- In a graph G = (V, E), we have

$$T(\mathbf{Bcast/RI+}) \geq max(d(x, y) : x, y \in V) = d$$

- Here **RI+** denotes **R** $\cup$ UI+

Worst case ideal time complexity

- `Flooding` performs broadcast in d ideal time units
- The lower bound is *tight*, i.e. it can be achieved
- `Flooding` is thereby *time optimal*

The ideal time complexity of ***Bcast*** *under* ***RI+*** *is* $\Theta(d)$

Message complexity

- We see that $M(\textbf{Bcast/RI+}) \geq n-1$
- We can, however, determine a more accurate lower bound
- Theorem 2.1.1:

$$M(\textbf{Bcast/RI+}) \geq m$$

Message complexity

- `Flooding` solves broadcast with $2m - n + 1$ messages
- This implies that $\mathrm{M}(\textbf{Bcast}/\textbf{RI+}) \leq 2m - n + 1$
- Combining with Theorem 2.1.1, we have

The message complexity of broadcasting under ***RI+*** *is* $\Theta(m)$

- `Flooding` is *message optimal* solution
- Improvements can affect only the constant factor; by Theorem 2.1.1 this cannot be reduced below 1

Broadcasting in Special Networks

- We have so far considered algorithms that are applicable to any network
- Recall that **R** contains the constraint CN, however
- We now explore algorithms that exploit structural knowledge of some special networks
- These algorithms are often more efficient, but lose generic applicability

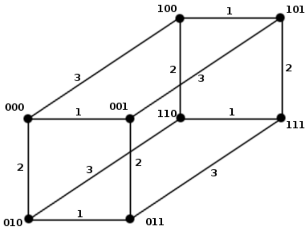
Broadcasting in Trees

- A tree is a graph that is connected and contains no cycles
- We know now that $m = n - 1$
- Hence the cost of `Flooding` is now $2m - n + 1 = 2(n-1) - (n-1) = n - 1$
- Note that this is true even when the entities don't know the network is a tree
- An interesting side effect is that the network becomes *rooted* in the initiator

Broadcasting in Oriented Hypercubes

- An oriented hypercube H_k of dimension $k = 1$ is just a pair of nodes called "0" and "1", connected by a link labeled "1"
- An oriented hypercube H_k of dimension $k > 1$ is obtained by taking two hypercubes of dimension $k - 1$, namely H'_{k-1} and H''_{k-1}, and connecting the nodes with same names with a link labeled k
- The name of each node in H'_{k-1} (respectively H''_{k-1}) is then modified by prefixing it with the bit "0" (respectively "1")

Broadcasting in Oriented Hypercubes



A hypercube H_k of dimension k has the following properties:

- Number of nodes $n = 2^k$
- Each node has exactly k links
- Number of links $m = nk/2 = (n/2)\log n = O(n \log n)$

Broadcasting in Oriented Hypercubes

- `Flooding` costs
  $2m - n + 1 = n \log n - n + 1 = n \log n/2 + 1 = O(n \log n)$
  messages
- However, hypercubes are highly structured networks, and we can exploit this knowledge to create a more efficient protocol

Broadcasting in Oriented Hypercubes

Strategy `HyperFlood`:

1. The initiator sends the information to all its neighbors
2. A node receiving a message on a link labeled l will only send messages to links with label $l' < l$

`HyperFlood` costs:

- $\mathbf{T}[\texttt{HyperFlood}/H_k] = k$
- $\mathbf{M}[\texttt{HyperFlood}/H_k] = n - 1$

Broadcasting in Oriented Hypercubes

Summarizing, we have:

The ideal time complexity of broadcasting in a k-dimensional hypercube with a dimensional labeling under ***RI+*** *is* $\Theta(k)$.

The message complexity of broadcasting in a k-dimensional hypercube with a dimensional labeling under ***RI+*** *us* $\Theta(n)$.

Broadcasting in Complete Graphs

- A complete graph is a graph where every node is directly connected to every other node
- As there are $m = n(n-1)/2$ links, `Flooding` would require $O(n^2)$ messages

Broadcasting in Complete Graphs

Strategy `KBcast`:

1. The initiator sends the information to all its neighbors

The ideal time complexity of broadcasting in a complete graph under RI+ is $\Theta(1)$.

The message complexity of broadcasting in a complete graph under RI+ is $\Theta(n)$.

Wake-Up: The Problem

- Consider a distributed computing system, where a task must be performed in which all the entities must be involved; however, only some of them are independently active and ready to compute
- To perform the task, we must ensure that all entities become active. This problem is called Wake-Up
- Broadcast is a special case of Wake-Up where there is only one initiator

Flooding for Wake-Up

Status Values: $S = \{\texttt{ASLEEP}, \texttt{AWAKE}\}$
$S_{\texttt{INIT}} = \{\texttt{ASLEEP}\}$
$S_{\texttt{TERM}} = \{\texttt{AWAKE}\}$
Restrictions : **R**

ASLEEP
Spontaneously
begin
send(W) **to** N(x);
become AWAKE;
end

Receiving(W)
begin
send(W) **to** N(x) - {**sender**};
become AWAKE;
end

Flooding for Wake-Up

- The flooding strategy solves the more generic Wake-Up problem
- For message cost, we have $2m \geq \mathbf{M}[\texttt{WFlood}] \geq 2m - n + 1$
- Message complexity is usually higher than Bcast: $M(\textbf{Wake-Up}/\textbf{R}) \geq M(\textbf{Bcast}/\textbf{RI+})$
- But ideal time complexity usually less: $T(\textbf{Wake-Up}/\textbf{R}) \leq T(\textbf{Bcast}/\textbf{RI+})$

Flooding for Wake-Up

Summarizing the complexity of generic Wake-Up:

The ideal time complexity of Wake-Up under $\boldsymbol{R}$ *is* $\Theta(d)$.

The message complexity of Wake-Up under $\boldsymbol{R}$ *is* $\Theta(m)$.

Wake-Up in Trees

- Message cost depends on the number of initiators, denoted $k_\star$
- Note that $k_\star$ is not a system parameter (but is bounded by one: $k_\star \leq n$)
- $\mathbf{M}[\texttt{WFlood}/\text{Tree}] = n + k_\star - 2$

Wake-Up in Labeled Hypercubes

- With BCast, we managed to create a solution (`HyperFlood`) for hypercubes that had message cost $O(k)$
- Unfortunately, with multiple initiators this is not possible
- We might as well just use `WFlood`, which uses $O(n \log n)$ messages

The message complexity of Wake-Up under $\boldsymbol{R}$ *in a k-dimensional hypercube with dimensional labeling is* $\Theta(n \log n)$

Wake-Up in Complete Graphs

- `WFlood` message cost is $O(n^2)$ in complete graphs
- `KBcast` would use $k_\star(n-1)$ messages
- This is again the best we can do, and we have

The message complexity of Wake-Up in a complete graph under $\boldsymbol{R}$ *is* $\Theta(n^2)$.

Wake-Up in Complete Graphs with ID

- To improve performance, we further assume the restriction Initial Distinct values (ID)
- This means each entity has a unique name
- Under these restrictions, algorithms have been devised that solve Wake-Up with $O(n \log n)$ messages
- These algorithms solve also the much more complex problem of *Election*

Traversal: The Problem

- Consider a distributed computing system, where each of the entities must perform some action, but in such a manner that at any given time only one entity at a time is active. This problem is called Traversal.
- Traversal is a *sequential* protocol
- Traversal is solved by letting a *traversal token* (or just token) to reach every entity sequentially
- Once a node has been reached, it is marked as "visited"

Depth-First Traversal (**DFT**)

Strategy `DFTraversal`:

1. When first visited, an entity remembers who sent the token, creates a list of all its still unvisited neighbors, forwards the token to one of them (removing it from the list), and waits for for its reply return the token
2. When the neighbor receives the token, it will return the token immeadiately if it has been already visited by somebody else, notifying that the link is a backedge; otherwise, it will first forward the token to each of its unvisited neighbors sequentially, and then reply returning the token

Depth-First Traversal

3. Upon the reception of the reply, the entity forwards the token to another unvisited neighbor

4. Should there be no more unvisited neighbors, the entity can no longer forward the token; it will then send the reply returning the token to the node from which it first received it

Depth-First Traversal

- There are three different message types: *T*, *Return* and *Backedge*
- On each edge, exactly two messages are transmitted during the protocol
- Recall that traversal is sequential, hence we have

$$\mathbf{T}[\texttt{DFTraversal}] = \mathbf{M}[\texttt{DFTraversal}] = 2m$$

Complexity of Depth-First Traversal

- Theorem 2.3.1 and Theorem 2.3.2:

$$M(\textbf{DFT/R}) \geq m$$

$$T(\textbf{DFT/R}) \geq n - 1$$

- `DFTraversal`, with message cost $2m$ is message optimal
- But the worst case ideal time complexity of $2m$ is no good; $2m$ could be several order of magnitude higher than the **DFT** lower bound $n - 1$
- For example, in complete graphs $m = n^2 - n$

Hacking (Protocol Optimization)

- Since the time cost of DFTraversal is far from optimal, measures have been taken to develop optimized protocols that would achieve depth-first traversal faster
- When measuring ideal time, only synchronous execution need be considered; however, this is not the case when considering algorithm correctness
- To improve time usage, we must either reduce the number of messages or introduce some concurrency

Hacking (Protocol Optimization)

Basic Hacking ("DF+")

- Idea: Avoid sending messages to back-edges
- Solution: Notify neighbors upon visit and wait for acknowledgement
- $\mathbf{T}[\texttt{DF+}] = 4n - 2$, $\mathbf{M}[\texttt{DF+}] = 4m$

Advanced Hacking ("DF++")

- Idea: Avoid sending acknowledgement messages
- Solution: The protocol is still correct, but errant T messages occur
- $\mathbf{T}[\texttt{DF++}] = 2n - n$, $\mathbf{M}[\texttt{DF++}] \leq 4m - n + 1$

Hacking (Protocol Optimization)

Extreme Hacking ("DF*")

- Idea: Use T message as implicit "visited" notification
- Solution: Saves a few messages and some time units
- $\mathbf{T}[\texttt{DF*}] = 2n - 2$, $\mathbf{M}[\texttt{DF*}] = 4m - 2n + f_\star + 1$

The ideal time complexity of depth-first traversal under $\boldsymbol{R}$ *is* $\Theta(n)$.

Traversal in Trees

- In trees, there are no backedges, which helps
- $\mathbf{M}[\texttt{DFTraversal}] = \mathbf{T}[\texttt{DFTraversal}] = 2n - 2$
- As a side effect, traversal constructs a so-called *virtual ring* of the nodes

Traversal in Rings

- A ring is a graph where every node has exactly two neighbors
- Depth-first traversal is achieved by simply selecting a direction on the ring
- Message cost and ideal time are both n for this algorithm

Traversal in Complete Graphs

- Depth-first traversal is again achieved very simply by the initiator sequentially passing the token to all its neighbors
- Message cost and ideal time are both $2n - 2$ for this algorithm

Practical Implications

- Our study of these protocols and their efficiency indicates that costs are relative to the number of edges in the graph
- In practise, this would seem to imply that well connected graphs lead to inefficient usage
- We can circumvent this adverse result by a simple insight: We can operate on any subnet of G and ignore the rest of edges

Subnets

- Which subnet of G should we choose?
- For any connected, undirected graph we have $(n^2 - n)/2 \geq m \geq n - 1$
- In particular, the upper limit is true for complete graphs and the lower for trees
- This leads to the conclusion that we should choose a spanning tree
- More on spanning trees on the upcoming presentations!

Questions and Answers

Any Questions?

Questions and Answers

Thank You!