Algorithms and Distributed Computing

Presentation to CPSC 181
March 2009

Prof. Jennifer L. Welch
Parasol Lab
Department of Computer Science and Engineering
Texas A&M University
Outline

- Centrality of notion of algorithm
- What is an algorithm?
- Designing and analyzing algorithms
- Understanding lower bounds and impossibility results
- Distributed systems
- Some sample distributed algorithms
  - clock synchronization
  - routing based on link reversal
What is Computer Science?

Bierman: Computer science is the study of algorithms
- how to conceive them and write them down, programming-in-the-small vs. programming-in-the-large
- how to execute them (why does a machine act the way it does, what are limitations, what improvements are possible)
What is Computer Science?

- Brooksheer: "Computer Science is the discipline that seeks to build a scientific foundation for such topics as computer design, computer programming, information processing, algorithmic solutions of problems, and the algorithmic process itself."
  - Most fundamental concept of CS is an algorithm: a set of steps that defines how a task is performed
  - An algorithm is instantiated in a program and then executed on a machine
Brookshears's Diagram

- Limitations of Algorithm
- Execution of Algorithm
- Communication of Algorithm
- Representation of Algorithm
- Discovery of Algorithm
- Analysis of Algorithm

- Theory of computation, ...
- Architecture, operating systems, networks, ...
- Algorithmics, ...
- Software engineering, ...
- Artificial intelligence, ...
- Data structures, programming language design, ...
What is Computer Science?

Schneider and Gersting: Computer science is "the study of algorithms, including their formal and mathematical properties

1. their hardware realizations
2. their linguistic realizations
3. their applications"
Schneider & Gersting's Diagram

1. Algorithmic Foundations of CS
   - design & analysis of algorithms, …

2. The Hardware World
   - computer organization, …

3. The Virtual Machine
   - assemblers, operating systems, …

4. The Software World
   - programming langs, compilers, …

5. Applications
   - artificial intelligence, …

6. Social Issues
What is Computer Science?

- C.A.R. Hoare: the central core of computer science is "the art of designing efficient and elegant methods of getting a computer to solve problems"

- D. Reed: Identifies 3 main themes:
  - hardware: circuit design, chip manufacturing, systems architects, parallel processing
  - software: systems software (e.g., operating systems), development software (e.g., compilers), applications software (e.g., web browsers)
  - theory: understand inherent capabilities and limitations of different models of computation (for instance, proving that certain problems CANNOT be solved algorithmically)
What is an algorithm?
- a step-by-step procedure to solve a problem
- every program is the instantiation of some algorithm

http://blog.kovyrin.net/wp-content/uploads/2006/05/algorithm_c.png
Solving a general, well-specified problem
- Given a sequence of \( n \) keys, \( a_1, \ldots, a_n \), as input, produce as output a reordering \( b_1, \ldots, b_n \) of the keys so that \( b_1 \leq b_2 \leq \ldots \leq b_n \).

Problem has specific instances
- [Dopey, Happy, Grumpy] or [3,5,7,1,2,3]

Algorithm takes every possible instance and produces output with desired properties
- Insertion sort, quicksort, heapsort, …
Challenge

- Hard to design algorithms that are
  - correct
  - efficient
  - implementable on real computers

- Need to know about
  - design and modeling techniques
  - resources - don't reinvent the wheel
Correctness

- How do you know an algorithm is correct?
  - produces the correct output on every input

- Since there are usually infinitely many inputs, it is not trivial

- Saying "it's obvious" can be dangerous
  - often one's intuition is tricked by one particular kind of input
Tour Finding Problem

Given a set of $n$ points in the plane, what is the shortest tour that visits each point and returns to the beginning?

- application: robot arm that solders contact points on a circuit board; want to minimize movements of the robot arm

How can you find it?
Finding a Tour: Nearest Neighbor

- start by visiting any point
- while not all points are visited
  - choose unvisited point closest to last visited point and visit it
- return to first point
Nearest Neighbor Counter-example

works poorly here
How to Prove Correctness?

- There exist formal methods
  - even automated tools
- Even informal reasoning is better than none
- Seeking counter-examples to proposed algorithms is important part of design process
Efficiency

● Software is always outstripping hardware
  – need faster CPU, more memory for latest version of popular programs

● Given a problem:
  – what is an efficient algorithm?
  – what is the most efficient algorithm?
  – does there even exist an algorithm?
How to Measure Efficiency

• Machine-independent way:
  – analyze "pseudocode" version of algorithm
  – assume idealized machine model
    • one instruction takes one time unit

• "Big-Oh" notation
  – order of magnitude as problem size increases

• Worst-case analyses
  – safe, often occurs most often, average case often just as bad
A faster algorithm running on a slower machine will always win for large enough instances.
Modeling the Real World

- Cast your application in terms of well-studied abstract data structures

<table>
<thead>
<tr>
<th>Concrete</th>
<th>Abstract</th>
</tr>
</thead>
<tbody>
<tr>
<td>arrangement, tour, ordering, sequence</td>
<td>permutation</td>
</tr>
<tr>
<td>cluster, collection, committee, group, packaging, selection</td>
<td>subsets</td>
</tr>
<tr>
<td>hierarchy, ancestor/descendants, taxonomy</td>
<td>trees</td>
</tr>
<tr>
<td>network, circuit, web, relationship</td>
<td>graph</td>
</tr>
<tr>
<td>sites, positions, locations</td>
<td>points</td>
</tr>
<tr>
<td>shapes, regions, boundaries</td>
<td>polygons</td>
</tr>
<tr>
<td>text, characters, patterns</td>
<td>strings</td>
</tr>
</tbody>
</table>
Real-World Applications

- Hardware design, especially VLSI chips
- Compilers
- Routing messages in the Internet
- Architecture (buildings)
- Computer aided design and manufacturing
- Encryption
- DNA sequencing
- ...
What is a Distributed System?

- A collection of independent computing entities that communicate with each other to solve tasks
- Examples:
  - the Internet
  - a local area network in the CS department or your home
  - a multicore machine
  - sensor networks, mobile networks, vehicular networks,...
2. This generic example of a multicore SoC shows both the complexity of the devices and the programming challenges. In a hypothetical transition from a 130-nm SoC with a single processor to a multicore implementation at 65 nm, designers would have roughly four times as many transistors to work with.
Distributed Systems

- Distributed systems have become ubiquitous:
  - share resources
  - communicate
  - increase performance
    - speed
    - fault tolerance

- Characterized by
  - independent activities (concurrency)
  - loosely coupled parallelism (heterogeneity)
  - inherent uncertainty
Uncertainty in Distributed Systems

- Uncertainty comes from
  - differing processor speeds
  - varying communication delays
  - (partial) failures
  - multiple input streams and interactive behavior
Reasoning about Distributed Systems

- Uncertainty makes it hard to be confident that system is correct

- To address this difficulty:
  - identify and abstract fundamental problems
  - state problems precisely
  - design algorithms to solve problems
  - prove correctness of algorithms
  - analyze complexity of algorithms (e.g., time, space, messages)
  - prove impossibility results and lower bounds
Synchronizing Clocks in a Distributed System

- Each computer has its own hardware clock
  - used to measure the duration of time intervals
  - usually some small rate of drift away from real time
- Each computer adds some value to the hardware clock to get its logical clock
  - try to keep logical clocks together
  - try to keep rate of logical clocks approximately that of the hardware clocks
Measuring Clock Differences

- How to evaluate how close together clocks are?
- **Skew:** how far apart clock times are at a given real time, or
- **Precision:** how far apart in real time clocks reach same clock time
- These are the same when there is no drift...
Skew and Precision

- \( AC_i \)
- \( AC_j \)

- clock time
- real time

- T
- t

- skew
- precision
Synchronizing Clocks

If hardware clocks don't drift, then once clocks are adjusted, they stay the same distance apart.

Achieving $\varepsilon$-synchronized clocks:
- initially clocks are not close together
- computers exchange some information and adjust their logical clocks so that the maximum skew is $\varepsilon$
Bounded Message Delays

- Consider the problem when computers communicate by sending messages to each other
  - e.g., the Internet
- Assume there are known bounds on how long a message can take to arrive:
  - at least $d - u$ time
  - at most $d$ time
  - $u$ is the uncertainty
Two Processor Algorithm

- Consider this simple algorithm:
- $p_0$ uses its hardware clock as its logical clock
- $p_1$ adopts (its best estimate of) $p_0$’s logical clock as its logical clock
- How does $p_1$ do this? $p_0$ sends its clock time to $p_1$ in a message
- How to handle uncertain delay? Assume delay is in the middle of the range: $d - u/2$
Analysis of Two Proc. Algorithm

- What is the skew attained by the algorithm?
- If message really did take \( d - u/2 \) time to arrive, skew is 0 (best case).
- If message took \( d \) or \( d - u \) time, skew is \( u/2 \) (worst case).
- Can we do better, perhaps with a more complicated algorithm? *No.*
Proving Lower Bound on Skew

- It is possible to prove that NO ALGORITHM for this problem can achieve a better skew
  - in the worst case
  - under the same set of assumptions

- This is called a *lower bound* result, or *impossibility* result.
What About More Processors?

- What if we have more than two processors?
- What is the best skew achievable?
- For now, stay with our simple assumptions of no drift, no failures, and bounded delays
Star Algorithm for $n$ Processors

- Pick one proc (say $p_0$) and let every other proc try to adopt $p_0$'s clock using the 2-processor algorithm.
- Worst-case skew can be as large as $u$ (one proc is $u/2$ behind $p_0$'s clock and another is $u/2$ ahead)
Improved Algorithm for n Processors

- All processors exchange h/w clock values.
- Each processor estimates the difference between its own h/w clock and that of each other processor.
- Each processor computes the average of the differences and sets its adjustment variable to the result
Improved Algorithm

- Averaging algorithm can be proved to achieve worst-case skew of \((1 - 1/n)u\)
  - starts at \(u/2\) for 2 processors and then grows to almost \(u\)
- It can be proved that this is the best possible skew
  - under the given assumptions
Hardware clocks typically suffer from **drift** (gain or lose time).

Usually the drift is **bounded**, though.

For quartz crystal clocks, $\rho$ is about $10^{-6}$.
Other Wrinkles

- When clocks can drift, processors must continually resynchronize. Two problems:
  1. Establish: Get clocks close together.
- What if some of the processors are faulty?
  - crash, or
  - send out incorrect clock information
Other Wrinkles

- There are numerous algorithms and lower bounds relating to clock synchronization under various system assumptions
- For the Internet standard on clock synchronization, check out the Network Time Protocol (NTP)
Routing Messages in a Network

- Suppose you want to send a message to a computer that is not close to you.
- Use a routing service, which finds a path in the network to your destination.

http://www.uga.edu/~ucns/lans/tcpipsem/gateway.routing.example.gif
Routing in a Dynamic Network

- What if the layout ("topology") of the network changes?
- Need to find new paths to the destination
- How to do this in a distributed way?
  - without any one node having to know the entire network topology
Adapting to Topology Change

Arrows indicate preferred neighbor(s) for forwarding messages in order to reach D
Routing with Link Reversal

- distinguished destination node
- every communication link has a virtual direction
- ensure that every node has a directed path (w.r.t. directions on links) to destination
- if topology changes break this property, then nodes should be able to restore it
  - by reversing directions on some incident links
  - determine which ones in a distributed fashion
Two LR Routing Algorithms

- **Full Reversal (FR):**
  - when a node becomes a sink, it reverses all its incident links

- **Partial Reversal (PR):**
  - when a node becomes a sink, it reverses some of its incident links - those that have not been reversed since the last time the node was a sink

- Proposed by Gafni and Bertsekas
FR Example
PR Example
Implementation of FR

- Each node $i$ keeps a pair $(\alpha, i)$, where $\alpha$ is an integer; pair is called *height*.
- Link between two nodes is directed from node with higher height to node with lower height.
- At each iteration, if a node $i$ has no outgoing links, then set $\alpha$ to 1 greater than the maximum $\alpha$-value of all of $i$’s neighbors at the previous iteration.
"Implementation" of PR

- Try to reduce the number of link reversals by having a sink node reverse only some of its incident links.
- Use a triple $(\alpha, \beta, i)$ for the height, where $\alpha$ and $\beta$ are integers.
- At each iteration, if a node $i$ has no outgoing links, then, using the neighbors’ heights from the previous iteration, change $\alpha$ and $\beta$ a more complicated way
Use of Unbounded Counters

- Both the pair and the triple algorithms use unbounded counters: the $\alpha$ and $\beta$ components of the heights can grow without bound as the network keeps changing.
- This can be undesirable.
- Is it possible to achieve the same result with bounded counters?
- YES!
Our Contributions

- Novel formulation of FR and PR using only binary labels on the links
- Simple distributed algorithm for finding routes in acyclic graphs
- Identify sufficient conditions on initial labeling for correctness
- FR and PR are special cases
- Easy to state new algorithms
- Much simpler proof of correctness
**LR** Generic Algorithm

- Input is a directed acyclic graph with
  - distinguished node $D$
  - each link labeled with 0 ("unmarked") or 1 ("marked")
- while there exists a sink $v \neq D$ do:
  - if $v$ has an incident unmarked link then
    - reverse all incident unmarked links
    - flip the labels on all incident links
  - else // all of $v$'s incident links are marked
    - reverse all incident links // leave them marked
LR Example
Special Cases of LR Algorithm

- **Full Reversal:** Initially all labels are 1
  - Only ever execute LR2 step
  - All labels are always 1

- **Partial Reversal:** Initially all labels are 0
  - Execute both LR1 and LR2
  - Labels change
What about Performance of LR?

- Previous work studied
  - work: total number of reversals done by all nodes
  - time: total number of rounds, assuming maximum concurrency for sinks reversing of the pair and triple algorithms

- Results were of this form: for every $n$, there exists a graph with $n$ nodes in which at least one node does approximately $f(n)$ reversals / takes approximately $f(n)$ rounds.
<table>
<thead>
<tr>
<th>algorithm</th>
<th>time</th>
<th>work</th>
</tr>
</thead>
<tbody>
<tr>
<td>pair</td>
<td>$\Theta(n^2)$</td>
<td>$\Theta(n^2)$</td>
</tr>
<tr>
<td>triple</td>
<td>$\Theta(n \ a^* + n^2)$</td>
<td>$\Theta(n \ a^* + n^2)$</td>
</tr>
</tbody>
</table>

- time = number of iterations
- work = number of node reversals
- $n$ = number of nodes with no path to destination
- $a^* = \max \alpha - \min \alpha$ in initial state
Our Contributions

- Our formulation allows us to express the *exact* number of steps taken by *any node in any graph* in the generic algorithm.
- Expression depends only on the input graph.
- Has simple formulas when specialized to FR and PR.
- Exact formula helps in finding best and worst topologies.
Number of reversals by node $v$ equals number of links directed away from $D$ in the chain to $v$. Quantity decreases by 1 when $v$ takes a step.
Work Complexity of FR

**Theorem:** Number of steps taken by $v$ in FR is min, over all chains between $v$ and $D$, of number of links directed away from $D$.

$\min(2, 1) = 1$
Summary

- Algorithms are at the heart of computing
- It is important and challenging to analyze them for correctness, performance, and optimality
- Distributed systems are all around us
- The uncertainty in distributed systems adds to the challenges for analyzing algorithms
- There are lots of fascinating questions in distributed computing that require algorithmic solutions
References

- What is computer science:

- Introduction to algorithms and their analysis:

- Distributed systems:
References

- **Clock synchronization algorithms:**

- **Link reversal routing algorithms:**
  - B. Charron-Bost, A. Gaillard, J. Welch and J. Widder, "Routing Without Ordering," submitted for publication