Distributed Algorithms — A Rapid Overview

- Essential to have some awareness of:
  - algorithms used in distributed systems
  - data structure implications of distribution
  - factors that influence algorithm behaviour
- Examples of specific algorithmic issues:
  - problems unique to distributed systems
  - additional complications in distributed systems
  - new performance metrics
- Learning Objectives:
  - awareness of some interesting groups of algorithms
  - insight into specific problems that must be addressed
  - knowledge of some specific algorithms and their use
  - ability to select appropriate solutions
- Also must be aware of the impact of underlying system properties
  - e.g. reliable FIFO comms channels simplify many problems

Algorithmic Metrics

- Assessing the usefulness of an algorithm may involve performance estimates: classically space and time complexity
  - e.g. Sorting can be done in $O(n \log(n))$ time
- In distributed systems, there are more issues, including:
  - space: local, total, average, worst
  - time: local, total, end-to-end
  - comms: # messages, # bits, link loadings, traffic patterns
- For example:
  - the achievable level of concurrency may vary
    - reducing end-to-end time but increasing total time
  - global knowledge may be used
    - held in one place this will
      - reduce space costs, increase communication costs and time
    - held in many places this may
      - increase space costs, but reduce time and comms costs
    - unless it changes often, then maintaining consistency costs
      - in communications, and possibly on other metrics

Example Problem: Disseminating Information

- Classic problem in distributed computing
  - wish to send data from one component to some/all others
- Variety of solutions, very different styles:
  - Broadcast (brute force)
  - Exploiting system topology (existing structure)
  - Gossip (probabilistic; explicit or adventitious)
  - Constructing useful mechanism (created structure)
- Assumptions, for example:
  - System consists of a connected graph of:
    - nodes (processors)
    - undirected edges (bidirectional communication links)
  - Message based vs RPC based communication

Message-based brute force Broadcast

- Initiator:
  - Send message to every neighbour
- All other nodes:
  - On receipt of a message:
    - If it's the first time seen:
      - Send message to all neighbours (except sender)
    - otherwise:
      - ignore the message
- Observations:
  - Without the "first time" rule, this never terminates
  - The initiator doesn't know when the algorithm terminates
  - The message travels along all the comms channels
    - usually twice! though the "except sender" rule helps
  - Have to retain a list of "seen messages"
  - Cannot send the same message twice!

Exploiting Existing Structure I

- Suppose the nodes are organised into a ring structure:
  - Initiator:
    - Send message to "next" node
    - Await message from "previous" node
  - All other nodes:
    - On receipt of a message:
      - send message to "next" node
- Terminates, initiator is aware of termination, minimal message passing, minimal space requirements, can resend a message providing originator can distinguish their message
- In a complete graph:
  - initiator gives a copy to every neighbour

Exploiting Existing Structure II

- In a rectangular lattice:
  - Initiator:
    - Send message to up, down, left and right nodes
  - All other nodes:
    - On receipt of a message
      - from right: send it to the left
      - from left: send it to the right
      - from below: send it up, left & right
      - from above: send it down, left & right
  - Some parallelism, no need to remember messages
- In a torus:
  - could exploit parallelism as above, but must beware wrapping
  - could send out messages as above, but with a lifetime/hopcount of up to half the dimension vertically, and then half horizontally
  - could simply traverse the data structure one element at a time

Gossip/Epidemic Approaches

- Can use more relaxed approaches to dissemination
- Exchange information with *some* neighbours
  - as part of other communications
  - deliberately chosen (maybe at random)
- Information slowly spreads
- Probabilistic effects:
  - some of the nodes have the info after a given amount of time
  - a particular node may have the info at a particular time
- Low cost, very effective, no rigid guarantees

Created Structure

- Would like to only send N messages to reach N nodes
- Would like to cope with an arbitrary graph structure
- Would like some sense of time required to disseminate info
- Can create structure to achieve this (c.f. overlays in P2P)

Connected Graphs and Spanning Trees

- Given a connected graph
- A spanning tree is a subgraph that:
  - is connected
  - contains all of the nodes/vertices
  - contains the smallest possible number of edges
  - (hence it is a tree)

Constructing a Spanning Tree

- Initiator:
  - send messages to all neighbours and await replies
- All other nodes:
  - on receipt of first message:
    - note edge it arrived along, parent in spanning tree is at other end of that edge
    - send messages to all other neighbours and await replies
      - note whether or not we are accepted as their parent
    - when all replies received send our reply to parent telling them they are our parent
  - on receipt of subsequent messages:
    - immediately send reply back saying they are not our parent
- When initiator receives all replies, a spanning tree has been constructed
- The tree is rooted and directed
  - the "parent" relation points towards the root/initiator
- Any node can exploit the tree by ignoring the "direction" of links

Using a Spanning Tree I

- Can efficiently broadcast information:
  - source: send message to all neighbours in tree
    - i.e. pass the message to your parent and all children
  - all other nodes: on receipt of a message, pass it to all neighbours in tree except the sender
- Can also efficiently acquire information:
  - use a broadcast to trigger a *convergecast*

Convergecast in a spanning tree:

- Triggering node:
  - collect messages until all neighbours have communicated
  - consolidate received data plus own data
- All other nodes:
  - collect messages until all neighbours except one have communicated
  - consolidate received data plus own data
  - pass combined data to remaining neighbour

Using a Spanning Tree II

- Putting the pieces together:
  - If there isn't a spanning tree, initiate the building of one
  - Use the spanning tree thus:
    - Broadcast a 'convergecast trigger', to acquire data
      - i.e. act as initiator in a broadcast, where the message is: "do a convergecast, passing in your data"
    - Act as Triggering Node in the ensuing convergecast
    - Broadcast the outcome of the data processing
- Properties:
  - $O(E)$ messages & total time to build tree, $O(N)$ to use it
  - end-to-end time proportional to height of tree
  - tree shape determined by comms latencies
    - links reached first end up in the tree
  - not at all fault-tolerant
  - must beware confusion if multiple trees are constructed simultaneously, multiple broadcasts overlap etc

Example Problem: Global Snapshot

Want to determine some "global" value?

- Can ask everyone to note their data at a specified time
  - needs highly synchronised clocks & know in-transit msgs
- Can use broadcast-convergecast three times to: pause all activity, determine value, resume activity

With FIFO channels can use Chandy-Lamport algorithm:

- Initiator:
  - record local state and send *snap* messages out on all channels
  - on each channel:
    - record all messages received until a *snap* appears
- All other nodes:
  - on receipt of first *snap* message:
    - record local state and send *snap* messages out on all channels
    - on each channel:
      - record all messages received until a *snap* appears
        NB: channel used by initial *snap* is deemed empty

Example Problem: Synchronising Clocks

- Maintaining accurate clocks is a consensus problem
- Issues: initialisation, clock drift, required accuracy
- Accuracy:
  - Radio delay across UK is about 0.003 seconds
  - GPS receivers can get time accurate to a milisecond or better
  - Often, but not always, good enough

Basic Techniques:

- Assuming one machine has accurate time, ask it
  - problem: communication latency
- As above, but record duration of call and take midpoint
  - problem: assumes call/return symmetry; consistent slow drift
- Collect several such estimated clock times and average them

Network Time Protocol

- Goals:
  - accurate synchronization to UTC for Internet
  - reliability
  - scalable, with frequent resynchronisation
  - protection from interference
- Basic idea: hierarchical approach
  - Top levels of "tree": accurate (atomic) clocks
  - Primary servers are trusted, accurate etc
  - Secondary servers use statistical analysis to interpret results of communications with primary servers
  - Local area networks exploit basic techniques above

Logical and Vector Clocks

- Don't really want perfectly synchronized clocks
- Do need clocks to be synced reasonably well
- e.g. distributed make
  - must get the actions in the right order
- Logical clocks in each node & message: cheap & effective
- Lamport Logical Clock
  - a counter, incremented on each action/message send
    - inc max of own clock and message clock on receipt
- Vector Clocks
  - a vector of counters at each node, with one slot per node
    - inc own counter in local vector on action/message send
    - take max of all vector elements on message receipt
- Lamport logical clocks embody some aspects of causality
- Vector clocks embody more causal ordering knowledge

Problem Solving in Grid Systems

- Usual issues:
  - organise data and computation
    - what to do
    - how to do it
    - when to do it
- Additional complication in distributed systems:
  - where to do it
    - where is the data
    - where will the processing take please
  - move data to the computational elements?
  - move computations to the data?
- Major new issue for Grid computing:
  - the components belong to autonomous organisations
    - nodes, data, code, etc
  - need to agree on security etc and compromise on locations