Introduction

29 July 2020

1. Organisation
2. Distributed algorithms
3. Graphs and spanning trees revisited
4. Ba
5. Ec
6. Ec
7. Echo/size algorithm
8. Further algorithms
9. Project and practical work
10. Readings

Organisation

- For additional details, see the DCO and the Canvas Syllabus
  - https://courseoutline.auckland.ac.nz/dco/course/COMPSCI/711/1205

    https://canvas.auckland.ac.nz/courses/45914

- A bigger assignment, called project plus re

- Exam: theory of the distributed algorithm
- Assignments: practical implementations, emulations of specific distributed algorithms on a single computer

Distributed algorithms

- Computing: computer systems, networking, computer architectures, computer programming, algorithms
-
- Distributed computing: distributed syst communication channels, distributed a programming, distributed algorithms
  - concurrency of components
  - paramount messaging time
  - lack of global clock in the async case
  - independent failure of components

Overlap between parallel and distributed computing

- One litmus test:
  - Parallel computing: tight coupling between tasks, that cooperate on shared memory

    Distributed computing: loose coupling between nodes, that

  - In classical algorithms, the problem is encoded and given to the (one single) processing element

  - In parallel algorithms, the problem is enc and then its chunks are given to the process
  - In distributed algorithms, the problem is given by the network itself and solved by coordination protocols
- More:

https://en.wikipedia.org/wiki/Distributed_computing#Parallel_and_distributed_computing

Typical scenario in distributed computing

- computing nodes have local memories and unique IDs
- nodes are arranged in a network: graph, digraph, ring, . . .
- neighbouring nodes can communicate by message passing
-
- the network topology (size, diameter, nei other characteristics (e.g. latencies) are o individual nodes
- the network may or may not dynamically change
- nodes solve parts of a bigger problem or have individual problems but still need some sort of coordination – which is achieved by distributed algorithms

Recall from graph theory

- Graphs, edges, digraphs, arcs, degrees, connected, complete graphs, size, distance, diameter, radius, paths, weights/costs, shortest paths, spanning trees, ...

-

  - min
- diameter = maximum distance, betwee

  - max min
- radius = minimum maximum distance, for any node to any other node (minimum attained at centers)
  - min max min

Typical graphs notations

Graph: (V, E)

- V, set of nodes (vertices), $N = |V|$
-
-
- node eccentricity = longest geodesic dista
  shortest path) from this node to other nodes

- D, diameter = maximum eccentricity, ov
- R, radius = minimum eccentricity, from any node
- centre (node) = node with minimum eccentricity (not unique)

Geodesics examples

- Three distinct spanning trees (rooted at 1) on the same undirected graph
-
-

BFS and DFS spanning trees

- Reflexive edge: an edge that loops back to the same node
- Default assumption: unless specifically said so, graphs are irreflexive (no reflexive edges)
-

- BFS spanning tree characterisations
  - Each node at graph hop distance d in the tree
  - Frond edges only between nodes at dept one (thus linking nodes on different branches)
- DFS spanning tree characterisation
  - Frond edges only between between nodes on the same branch (a frond links an ancestor with a descendant)

BFS and DFS spanning trees – examples

- Three distinct spanning trees (rooted at 1) on the same undirected graph
- Which one is BFS or DFS (start from root node 1)?

- Left is BFS; Right is DFS (when we first go to node 2)
- Middle is neither, but could be BFS, if we start from 2...
- ... or DFS, if we start from 3 or 4

Rounds and steps

- Nodes work in rounds (macro-steps), which have three

    3. Send sub-step: send outgoing message

- Note: some algorithms expect null or empt explicit confirmation that nothing was sent

Timing models

1. Synchronous models
   - nodes work totally synchronised – in lock-step
   - easiest to develop, but often unrealistic and less efficient

2.
   - often unrealistic and sometimes impos

3. Partially synchronous models
   - some time bounds or guarantees
   - more realistic, but most difficult

- Quiz: guess which models have been first studied?
  Heard of Nasreddin Hodja's lamp? ☺

Synchronous model – equivalent versions

- Synchronous model – version 1
  - all nodes: process takes 1 time unit
  - nits

-
  - all nodes: process takes 0 time units
  - all messages: transit time (send

- The second (and equivalent) version ensures that synchronised models are a particular case of asynchronous models

Asynchronous model

- Asynchronous model

  - all nodes: process takes 0 time units

    each message (individually): transit time (send receive)

  - often, a FIFO guarantee, which howeve

    timing assumption (see next slide)

- Time complexity (worst-case) = supremum of all possible normalised async runs
  - NOTE: async time complexity ≥ sync time complexity as the sync run is just one of the all possible runs

Asynchronous model

- Asynchronous model with FIFO channels
  - the FIFO guarantee: faster messages cannot overtake earlier slower messages sent over the same channel

    congestion (pileup) may occur and this should be accounted for

    delivered after an additional arbitrary d ..
    [Lynch]
  - thus, a sequence of n messages
    last is delivered
  - essentially, a FIFO "channel" may not be a simple channel, but need some middleware (to manage the message queue)
  - suggestion to develop robust models, who do not rely on any implicit FIFO assumption [Tel]

Nondeterminism

- In general, both models are non-deterministic

- However, all executions must arrive to a vali
necessarily the same, but valid)
  - If always same decisions, then the system is called confluent

Starting and stopping options

- Starting options
  - Single-source or centralised or diffusing: starts from a designated initiator node

- Termination options
  - Easy to notice from outside, but how do the know this?
  - Sometimes one node (often the initiator) takes the final decision and can notify the rest (usually omitted phase)
  - In general, this can be very challenging and requires sophisticated control algorithms for termination detection

Echo algorithm

- Echo is a fundamental diffusing (single source) algorithm, which is the basis for several others

- combines two phases (imagine a stone thrown into a pond

  - a convergecast, "bottom-up" or echo phase, which confirms the termination

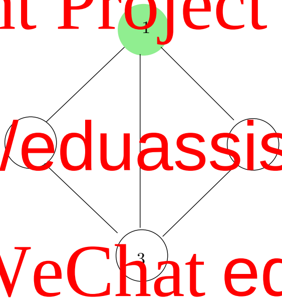
  - parent-to-child links can be built either a by additional confirmation messages i broadcast (not shown in the next slides)

- after receiving echo tokens from all its neighbours, the source node decides the termination (and can optionally start a third phase, to inform the rest of this termination)

Echo algorithm

- Echo algorithm is an instance of a large family known as wave
-
  BFS spanning tree – Echo is also known as SyncBFS
- In the async mode, the broadcast phase of Ec
  spanning tree, but not necessarily a BFS spa

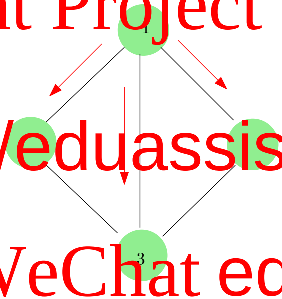
Echo Algorithm - Sync



Time Units = 0

Messages = 0

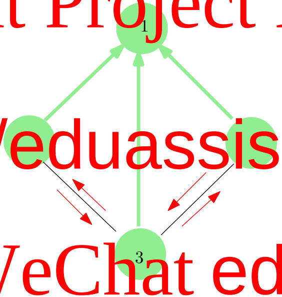
Echo Algorithm - Sync



Time Units = 1

Messages = 3

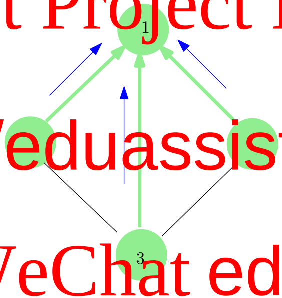
Echo Algorithm - Sync



Time Units = 2

Messages = 7

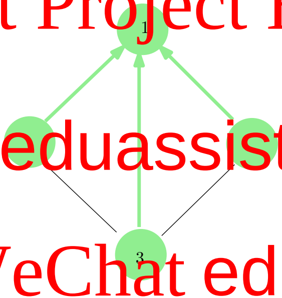
Echo Algorithm - Sync



Time Units = 3

Messages = 10

Echo Algorithm - Sync



Time Units $= 3 \leq 2D + 1$

Messages $= 10 = 2|E|$

Echo programs (Tel)

- Each node has a list, Neigh, indicating its neighbours

- There are two different programs: (1) for the initiator, and (2)

-
  reduced to this)

- With the exception of the initiator's first ste
  active only after receiving at least one messa
  idle (passive) between sending and receiving new messages

- Exercise: try to translate the following pseudocodes into a state machine format

Echo program for initiator (Tel)

```
let parent = null
let rec

for [illegible]
    send tok

while rec < |Neigh| do
    receive tok
    rec += 1

decide
```

Echo program for non-initiators (Tel)

```
let parent = null
let rec = 0

receive tok
parent = q
rec

for q in Neigh \ parent do
    send tok to q

while rec < |Neigh| do        // cou
    receive tok               // forward and return
    rec += 1

send tok to parent
```

Sync vs. Async

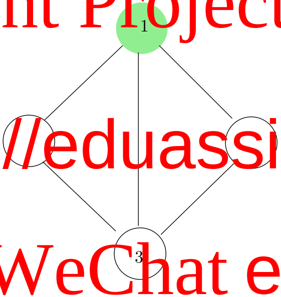
- Using the informal complexity measures appearing in the preceding slides (there are others), we conclude

- Sync: time complexity = $O(D)$, message complexity = $O(|E|)$
-

- However, the runtime complexity measu (drastically)

- Async: time complexity=$O(|V|)$, $|E|)$
- Why?
- For the time complexity, we take the supremum over all possible normalised executions (delivery time in $[0, 1]$)

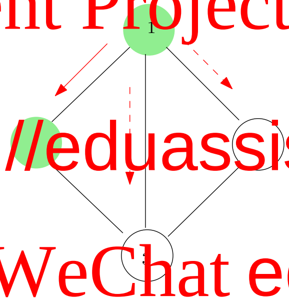
Echo Algorithm - Async



Time Units = 0

Messages = 0

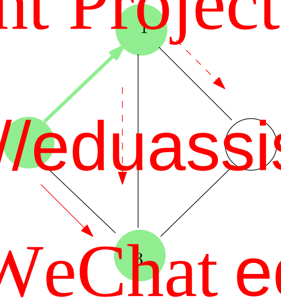
Echo Algorithm - Async



Time Units = ε

Messages = 3

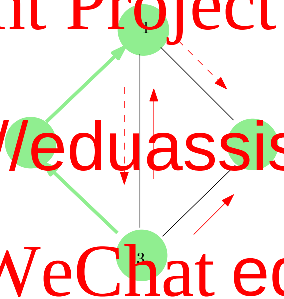
Echo Algorithm - Async



Time Units $= 2\varepsilon$

Messages $= 4$

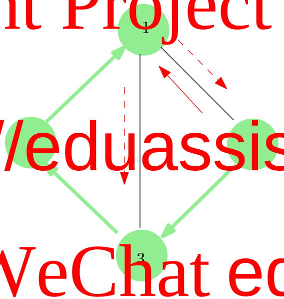
Echo Algorithm - Async



Time Units = 3ε

Messages = 6

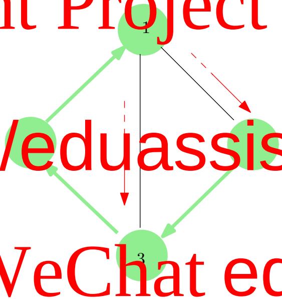
Echo Algorithm - Async



Time Units = 4ε

Messages = 7

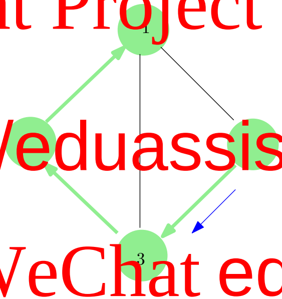
Echo Algorithm - Async



Time Units = 1

Messages = 7

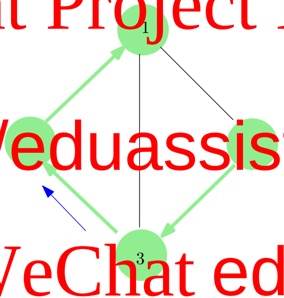
Echo Algorithm - Async



Time Units = 2

Messages = 8

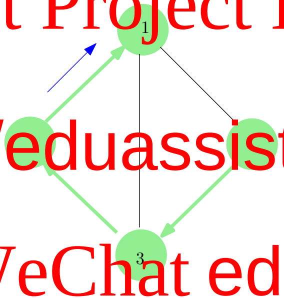
Echo Algorithm - Async



Time Units $= 3$

Messages $= 9$

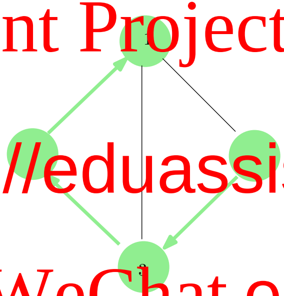
Echo Algorithm - Async



Time Units = 4

Messages = 10

Echo Algorithm - Async



Total Time Units = 4

Time Units on Broadcast = 1

Time Units on Convergecast = 3 = $|V| - 1$

Messages = 10

Echo algorithm revisited

- Like other members of the wave algorithm family, Echo can be adapted to determine:

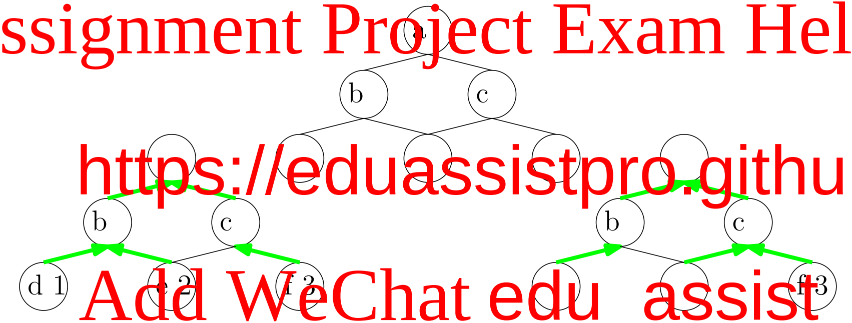
  - The size of the network (number of nodes)

  - In general, functions which are associati such as + (why?)

  - A "leader" (e.g. the node with the highest I

- A simple "trick": values can be attached to the tokens!
  - Forward token with null/zero value
  - Return token to parent with subtree value

Echo/size program – associativity and commutativity



Non-determnistic but confluent evaluations – all return 6!

- Left: (1+2)+3, (2+1)+3, 3+(1+2), 3+(2+1)
- Right: 1+(2+3), 1+(3+2), (2+3)+1, (3+2)+1

Echo/size program for initiator

```
let parent = null
let rec = 0
let

for
    send (tok,0)

while rec < |Neigh| do
    receive (tok,s) // order irrel
    rec += 1
    size += s

decide size
```

Echo/size program for non-initiators

```
let parent = null
let rec = 0
let size = 1

receive (tok,s)
parent = q
rec [illegible]

for q in Neigh \ parent do
    send (tok,0) to q

while rec < |Neigh| do
    receive (tok,s)            // fan-out tokens: s=0
    rec += 1                   // fan-in tokens: s=subtree size
    size += s

send (tok,size) to parent  // only children really contribute
```

Further algorithms

- Besides some basic fundamental algorithms, we intend to

  - Byzantine agreement: ”the crown jew algorithms” – Dijkstra prize 2005, 2001

  - The relation between Byzantine algori
  - all these have practical significance and applications

Project and practical work

- Practical work is necessary to properly understand the concepts

-

- better, emulate distributed systems on a si

- For uniformity and fairness in marking, we u specifically with C# (or F#)
- The final exam is focused on concepts, does not contain "technological" questions (related to .NET)

Prerequisites (cf. 335)

- Familiarity with C#, at least to the level presented in the C# 5.0 Specifications, Chapter Introduction (pp 1–32):
-
- Familiarity with the specific API that we will distributed systems

- Familiarity with searching and reading th
- Familiarity with Linqpad `http://www.linqpad.net/`

How to emulate sync and async distributed systems?

-
-
- Multi-process emulation by HTTP/RE
  Sinatra)

Readings

- Textbooks (in library, also limited previews on Google books)
  - Lynch, Nancy (1996). Distributed Algorithms. Morgan

    978-0-52179-483-1.
  - Fokkink, Wan (2013), Distributed Alg

    Approach, MIT Press (Second Edition
- Research and overview articles (generally overlapping the textbooks) will be indicated for each topic

Readings 2

- My articles (in collaboration) on bio-inspired models for distributed algorithms may offer additional views

  - network discovery

  - image processing (stereo, skeletonisa

  - Byzantine agreement
  - formal verification
  - hard problems (SAT, TSP, QSAT)

- Pre-print versions published in the CDMTCS research reports

  `https://www.cs.auckland.ac.nz/staff-cgi-bin/mjd/secondcgi.pl?date`