Master 2 Info Ubinet+Option SI5

Distributed Algorithmic

Françoise Baude

Université Côte d'Azur

Polytech

baude@unice.fr

web site : LMS of UCA

Alg. App. Distrib. Comp. EIIN932

https://lms.univ-cotedazur.fr/course/view.php?id=14171

Sept. 2021

Chapter 1 : Introduction - concepts, election

Organization 2021-2022

- 3h Course+Exercises spread on slides each week
- 7 weeks of lesson
- Final on-paper exam (8th week): Nov 15, morning
- Home work almost each week -> accumulate intermediary marks
- You can ask all your questions by email between 2 courses

- Different teachers:
  - Ludovic Henrio (CR CNRS), Scale team at CNRS I3S laboratory, now ENS Lyon; Etienne Lozes (PR UCA), Scale team
- A huge amount of literature, some of it referenced on our site
  - Most important books are cited on the web site, some of them are on-line available [Tanenbaum, Ghosh, Coulouris, Raynal, Mattern, …] or at the Library (Campus SophiaTech)
  - Whenever relevant, pdf files detailing important research (background) results

Overall overview of the course (1/2)

1. Introduction and First simple algorithms [FB]
   - System model, assumptions, notations
   - Distributed Approximation problem
   - Leader Election
2. Time in distributed systems [FB]
   - Causality relation
   - Timestamps, Integer and Vector clocks
   - Consistent or non-consistent cuts
     - Illustration with termination detection
3. **Faults and Recovery [LH, online]**
   - Classification of faults
   - Fault tolerance by recovery
     - Checkpointing and rollback recovery
     - Message logging

Overall overview of the course (2/2)

4. Group communications [FB]
- Definition, Examples, Use-cases of group. Comm.
- Ordering (local, causal, total)
- Reliability
- Closed/Open groups
- Communication group middleware: services, examples

5. Distributed consensus – [FB]
- Consensus problem in presence of faults
  - impossibility result in an asynchronous messaging system
  - Solutions for synchronous messaging systems
- Introduction to Failure detectors

6. Mutual Exclusion [EL]
- Token based algorithms
- Timestamps based algorithms
- Mixing of both

7. Global state collection-Distributed transactions [FB]
- Termination detection
- Deadlock detection (Resource or message-based deadlock)
- Distributed transactions
  - Concurrency control and serializability
  - Atomic commit protocols

Course 1: plan

1. Motivating example
2. Assumptions about the distributed system
3. Algorithm description, distributed execution representation and complexity evaluation
4. Leader election algorithms
   - Motivation: a leader is a coordinator
   - Leader election problem: a consensus problem
     - Distributed approximation algorithm, without faults, on
       - *completely connected topology*
       - *uni-directional ring*
       - *any topology*
     - Round-based algorithm
     - Specific algorithm, assuming faults are detected

5

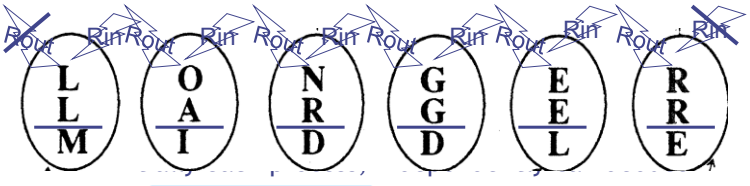
5

1. Motivation: distributed solution of a cryptarithmetic word puzzle [Mattern]

```
 LONGER
+LARGER
-------
=MIDDLE
```

PROBLEM: ASSIGN A VALUE ∈ [0, 9] AT EVERY LETTER

- Propose ≠ values for letters E.g.:



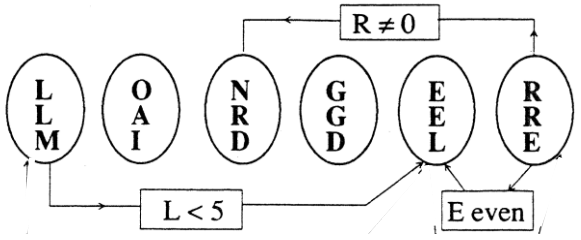
  - Rin on process P0 = 0 => nothing new
    - But, R + R + 0 => E is even, E in {0,2,4,6,8}
    - R ≠ 0, otherwise, E woud be 0 (forbidden 2+ letters equal)
  - Rout on process Pn = 0 => M < 10 (nothing new!)
    - But, even if Pn has no clue about its Rin, L+L+Rin <10 => L <5
- All corresponding messages are broadcasted
  - & are dropped by receivers in case their letters don't match

6

6

One *possible* distributed execution

- Pseudo phase 1



- On reception of E even on P1 => nothing new
- On reception of R ≠ 0 on P3 => nothing new
- On reception of L<5 on P1:
  - Try all possibilities for {(E,L)}: L≠0, otherwise E=L (in case Rin would be =0)

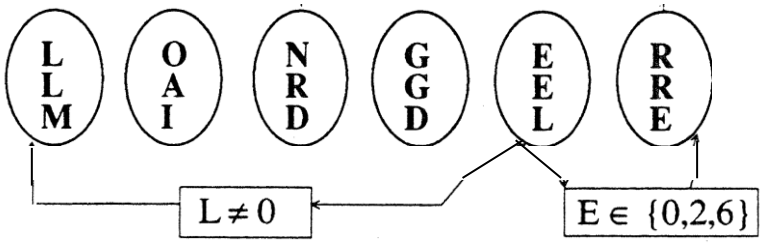
(0+0+Rin)={0,1} (but can't yet constraint Rin)
(2+2+Rin)={4,5} => E in {0,2,6} and
(4+4+Rin)={8,9}
(6+6+Rin)={12,13}
(8+8+Rin)={16,17}

7

7

One *possible* distributed execution

- Pseudo phase 2



- On reception of L≠0 on Pn:
  - Rout=0, but Rin?, L in {1,2,3,4} so =>

(1+1+Rin)={2,3}
(2+2+Rin)={4,5}
(3+3+Rin)={6,7}
(4+4+Rin)={8,9}

, M>1

- On reception of E in {0,2,6} on P0:
  - Rin=0, Rout?, R ≠ 0, so ….. => R in {1,3,5,6,8}

8

8

One *possible* distributed execution

- Pseudo phase 3

- Reception of messages triggered by phase 2 are received, but no new message is triggered
- So, the computation is passive (not dealocked, but nothing more happens)
  - We need to detect this situation, which is a **termination** situation (process passive& no mess.)
  - Once detected, a process (the **leader**, which may e.g. be P0) may be in charge of arbitrarily deciding of some constraints for some of the letters
    - R in {1,3,6} out of {1,3,5,6,8}
- Rem: messages are asynchronous and can be received in any non-deterministic order. We sketched one possible execution. An other could have triggered different messages.

How?

} To keep execution going on

2. Assumptions: distributed system

- Set of P0, P1,…, Pn process (CPU=site), with their own local memory, interconnected
  - MIMD principle
  - No global shared memory
  - No common clock
  - A process can send a message to any other process to which it is connected (has a reference to it)
    - These processes are its neighbors
  - Message transmission is (most frequently) reliable and either
    - Asynchronous (most frequent assumption)
    - Synchronous
    - Asynchronous, but with a known (bounded) delay
  - Point-to-point communication channels are FIFO
  - Processes are (most frequently) non-faulty

3. Algorithms: Model and Notations

- **Message driven**
- Each process runs its own algorithm, on its local state (=its local variables)
- Each algorithm is a set of guarded & atomic actions ("on event do:")
  - Internal actions are only triggered by condition on local variables

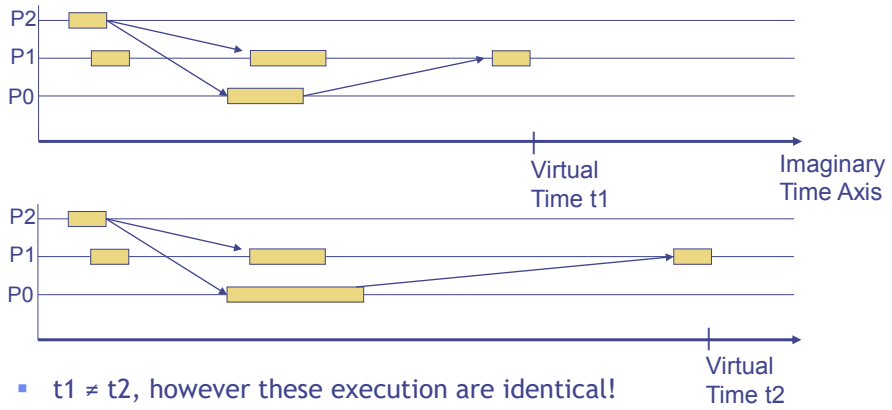
I_p: { guard }
sequence of statements which do only use the local variables;
send a message to ...;
sequence of statements which do only use the local variables;
...

  - Message-driven actions
    - Message received first
    - Guard can include condition on message tag, origin

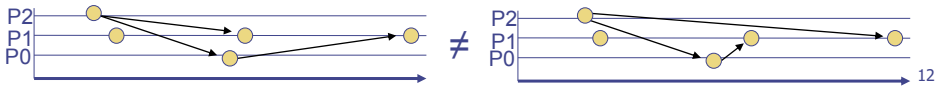
M_p: { guard }
receive message $\langle M \rangle$;
...

  - Non-determinism of choice between guards that are true

Execution representation: time diagram



- t1 ≠ t2, however these execution are identical!
- => Similar if we would only have Atomic, instantaneous actions
- What only matters: the order of message reception, whatever the transmission duration



Complexity evaluation

- Message transmission delay on any comm. channel >> computation speed on any proc.
- Evaluation of the algorithm
  - On one execution: one graph-based representation
    - Total number of messages exchanged
    - Depth of the associated graph: total time expressed in unit=delay to transmit one message on a comm. channel
    - Width: parallelism degree of the execution
  - On all executions
    - Best or Worst case: exhibit the associated graph among all possible graphs
    - Mean/average: reason upon all graphs

13

4. Leader Election

- Why a Leader: very useful to coordinate, orchestrate the execution of distributed applications
  - Go from a peer-to-peer, "democratic" organization
    - Where all peers are equal, play exactly the same role
    - Totally symmetric system
  - to a centrally controlled one (more easy to manage)
    - Where the leader has a specific and important role to play
      - I.e. some actions must be executed by only one process
        - Decide to pick a specific value for a letter in the puzzle ex.
        - In a clients-server architecture, the server is unique

14

Leader election problem=symmetry breaking problem

- Necessary for deciding which process among a set of equivalent processes must be the leader
  - Initially
  - Whenever the leader dies/fails
    - Election must be organized among all non-faulty processes
- Seek properties of the election algorithm
  - all non-faulty processes ends up reaching an agreement about who among them is the leader
    - It's not mandatory that all non-faulty processes are candidates
    - But all processes must participate, because at the end, they all must know who is the elected process

General view of the algorithm

- More or less well identified steps (algo. dependant):
  - (detection of the need of election)
  - Preparation
  - Election (of one candidate process among all candidates)
    - Candidates also vote, and we must determine when vote is terminated before …
  - … the Proclamation of the result
- Should be distributed
  - No central place for all processes to send their vote!
- Candidate values (identifiers)
  - Must be unique and totally ordered
    - E.g. <1/cpu load on site i, i>
    - Usually: value = process id, as we assume all are unique (e.g IP . Process Id), and leader = the highest value
    - There exists algorithms in case processes are anonymous (no id)!
  - Comparison function between 2 values is associative

4.a Election seen as a distributed approximation computation

- Each candidate value: an approximation of the final result!
- At any point in time, each process participating in the election only knows approximately who is the winner
- Election core step role: refine this knowledge

(0) Compute an initial approximation and distribute it to the other processes.
(1) Wait for a new partial approximation communicated by some other process.
(2) On receipt of such an information
  (a) combine it with the current local approximation and try to compute a better approximation,
  (b) if this yields a better approximation then distribute it to the other processes.
(3) Go to step (1).

17

Distributed approximation: underlying topologies

- Topology directly impacts the convergence speed, (due to total number of messages, and reception order)
  1. Totally connected: each process is directly connected to any other
     - Worst case: process 1 receives all messages from the N-1 others, in the increasing order of process ids
       - Learns that P2 is possible better winner -> N-2 new messages
       - Learns that P3 is possible better winner -> N-2 new messages
       - …
       - Learns that PN is possible better winner -> N-2 new messages
       - Total messages generated by P1 = O(N*N), **total → O(N*N*N)**
     - Best case: each process receives all messages in decreasing order of process ids
       - P1 learns that PN is possible better winner -> N-2 messages
       - P2 learns that PN is possible better winner -> N-2 messages
       - …
       - **Total messages = O(N*N);** //Time excluding proclamation = (N-1)

18

18

Chang & Roberts ring-based algorithm

- Intentionally restrict the communication capabilities: uses an embedded unidirectional ring for propagating election-related messages
  - Assume there exists such a ring of all (non-faulty) processes
  - Election on each Pp:
  - Proclamation is only immediate on P_{leader}
  - Complete Proclamation:
    T_p: {A message <End>arrivec
    if p≠M then
    send <End> to neighbor;
    leader=M;
  - Termination detection before proclamation can start is for free!
    - When leader receives back its value as an echo: confirmation that all have received leader value

$\mathbf{I}_p$: { $M = 0$ }
$M := p$;
send $\langle M \rangle$ **to** neighbor

$\mathbf{R}_p$: { A message $\langle j \rangle$ has arrived }
if $M < j$ **then**
$M := j$;
send $\langle M \rangle$ **to** neighbor;
fi;
if $j = p$ **then** *"I am the leader"* **fi**
Send <End> to neighbor;

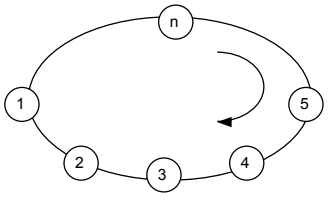
19

19

Chang&Roberts algorithm (core phase): complexity, n candidates

- Depends of the topology (=ring), but also, of the way process ids arise on the ring
  - This drives the extinction of candidates' messages
    - A message is not forwarded when it meets a higher-numbered process

Worst case:
The candidate message of each Pi, travels i times, before extinction. Only the winner message makes a complete turn. $\Sigma_1^n i = n*(n+1)/2$ messages



Best case:
The candidate message of each Pi, travels 1 time, before extinction. Only the winner message makes a complete turn. 2n messages

Average case:
computed on average on all possible ring organisations
O(n*logn) messages

"Echo" msg

//Time complexity: O(n-1+1) whatever ring organization=Ring traversal delay=Diameter

20

Chang&Roberts generalization [Tel]: total algorithms

- On Any topology (no partition), where all processes participate
  - Each process has some neighbors, say maximum E, bi-directional links
  - Diameter D
- Relies on "probe-echo", initiated from each candidate
  - each candidate process submits its value (probe) down to its rooted tree of (all) processes
    - Dynamic tree construction: father={from which I receive the value}; sons= {all neighbors – father}
  - The value is propagated down the tree only if it is a better approximation of the leader value
    - Otherwise, this execution of probe-echo dies, because not propagated
  - Only one probe-echo wave terminates
    - This is the one of the leader (maximum value)
    - It has to inform all processes that their Max value is actually the one of the leader !
- Exo: properly write this algorithm

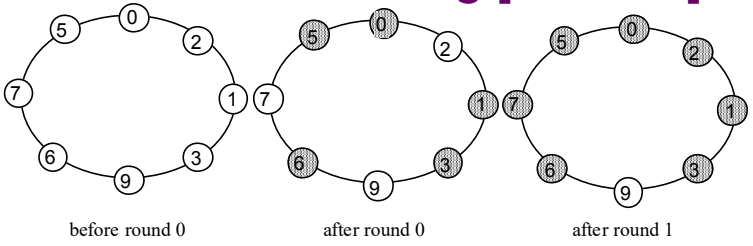
21

4.b Rounds-based election

- Principle: at each round, eliminate candidates
- On each Pi,
  - when (initially) white:
    - Send my value to my neighbors
    - Compare with values received from each neighbor:
      - eliminate myself as candidate (= turn black) if my value is <
      - If my value = all received values, "I'm the leader", proclaim result "finished, my value is the leader"
    - Pass to the next round if not finished
  - When black: only propagate messages
    - when I receive a message from a neighbor, I propagate it to my other neighbors

22

On a bi-directional ring [Franklin]



- The election needs at most O(log N) rounds
  - As at least half of white processes turns black at each round
    - Adjacent still-candidate values are compared: one of the 2 gets eliminated
- In each round: each of the N processes receives 2 messages, and forward them = 2N messages per round
  - //Time complexity=logN*(N-1) Total mess= 2N.logN at worst
- Exo: how many rounds and total messages
  - On one-directional ring: what are worst and best cases ?
  - On a full-connected topology: you can specialize the algo. (you know D=1 & proc number)
    Each process waits for one message from any other
    Then, it computes (sequentially) the maximum
    - N * (N-1) total number of messages

4.c Election Problem on a faulty system

- Communication links are fault-free, but, processes (including leader) can fail
  - Thanks to a timeout and fault-free comm. channel assumption (bounded synchronous)=> possible to detect if a process is faulty
- Assume a total connected topology, and *processes know the identity of all others* (quite strong assumption!)
  - When the leader is known to have failed, each process Pi spontaneously would like to replace it
  - Principle: bid that Pi is the new leader, which is true iff all higher-numbered processes have failed (i.e. they do not respond in the given timeout)

Bully algorithm [Garcia-Molina]

- (Detection phase): No answer when pinging the leader
- Preparation phase: Each Pi bids by sending a message "Election" to all j, j>i
- Election phase:
  - If no higher-numbered process respond within the *timeout*, Pi is the leader. Enter proclamation phase
  - If process Pj receives "Election" from Pi, i < j, it sends "Reply" to Pi, meaning "you can not be leader", and acts as a candidate (if not already) => go to preparation phase
  - If Pi receives "Reply", it knows it can not be the leader, waits for proclamation ("Leader" mess) to know who is the leader
- Proclamation phase:
  - If leader: send "Leader" to all j, j<i (rem: k>i have failed!)
  - If receiving message "Leader"=> Leader=identity of sender
  - If timeout expires, go back to Preparation phase (=probable leaders have failed in the meantime of the election)

Complexity, assuming n "simultaneous" candidates

- Each process Pi sends (n-i) "Election" messages
  - Process P1 (e.g. P1 detects first the leader dead) sends (n-1) mess, then P2, P3, …, Pn-1 (acting as a candidates) send messages to resp. P3, P4, …Pn. =$O(n^2)$ messages are generated in total
    - Each process Pj replies with "Reply" to potentially (j-1) processes
  - Pn replies to (n-1) messages, // time complexity = O(n-1)
- The would-be leader
  - sends "Leader" mess. to (i-1) processes
  - or fails: repeat the entire algorithm with one process less. At worst, repeat this algo. (n-1) times to end up with one elected Leader (among only one alive process!).
  - Worst case when n-1 failures during election=$O(n^3)$ message complexity

“Bully” behavior of the original version

- Rem: in French, bully means “brute, terreur, tyran”

- Arises when crashed processes are restarted after a failure, with the same identifier (the set of alive processes evolves)
  - The new process starts the bully algorithm
  - If it has effectively the highest process id. it proclaims itself as the leader (send the “Leader” msg)
    - Even if a new leader is in the process of being elected concurrently (between the crash and recovery); so we can get 2 leaders, and an inconsistent knowledge of who is the leader on processes.
- Exo:
  - run the algo. with p1, p2, p3, p4, assuming p4 has failed. During election, p3 fails, so p2 takes over the election and elects itself
  - p3 now recovers (or was slow but not failed=> timeout expired, as it was badly calibrated). Explain how it happens we have 2 leaders
    - We see that processes (e.g P1) can be the target of 2 subsequent messages “Leader” (one from P2, an other from P3). If it only takes the first that arrives (e.g from P3), and continue to run the application, then, P1 thinks leader = P3; After P2 has send “Leader” messages, it thinks leader=P2 (it might even not see the “Leader” message from P3, as it continues in the application).

27