A LOOK at DISTRIBUTED RECURSION

Michel RAYNAL

Institut Universitaire de France

and IRISA, Université de Rennes, France

Disclaimer

This talk is

- On distributed recursion
- Not on distributed systems

  e.g., Recursive tree traversal vs recursive structuring in OS

"Those who cannot remember the past are condemned to repeat it"

Reason in Common Sense (chapter 12, 1905), George Santayana (1863-1952)

Early papers: recursion, distributed comp./struct.

- Algorithms (dynamic)

  Lamport L., Shostack R. and Pease M.,
  The Byzantine Generals Problem.
  ACM Transactions on Programming Languages and Systems, 4(3)-382-401, 1982.

- Static design

  Randell B.,
  Recursively structured distributed operating systems.
  Proc. IEEE Symposium on Reliability in Distributed software and Database Systems, IEEE Press, pages 3-11, 1983.

Table of content

Part I

Recursion in Sequential Computing

Recursion in sequential computing (1)

- Strongly related to mathematical recurrence
- Operational point of view: Pushdown automata
- Intimately related to
  - ⋆ Specific data structures (e.g., tree traversal)
  - ⋆ "Divide and conquer" strategy + reflexivity
- Elegant formulation, inductive reasoning
- Invariant-based and induction-based proofs

- O.J. Dahl, E.W.D. Diskstra and C.A.R. Hoare, *Strucutred programing*, Academic Press, pp. 1972. (ISBN: 0-12-200550-3)

Recursion in sequential computing (2)

On the complexity side

- Logarithmic reduction $T(n) = \alpha T(n/\beta) + \gamma$ where $\beta > 1$
- But can be dangerous: exponential complexity
  - ⋆ When $T(n) = \alpha T(n - \beta) + \gamma$ where $\alpha > 1$
  - ⋆ But when computing data (Fibonacci, Binomial coef., matrix multiplication, etc.) we are saved by dynamic programming

Recursion in parallel computing

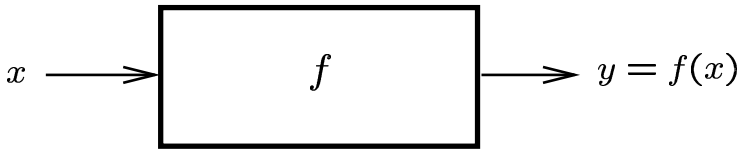
- No specific difference with sequential computing
- Exploit data independence (e.g., quicksort)
- Parallelism is looking for efficiency
  (Scheduling issues are crucial: SJF, etc.)

Part II

Sequential Computing vs Distributed Computing

Sequential computing

- Power and limit of sequential computing
- Notion of a function: $y = f(x)$



- Notion of a computable function
- Several formalisms:
  Turing machine,
  Post system,
  Church's lambda calculus, etc.

What is distributed computing about?

- Real-time: masters On-time computing
- Parallelism: provides Efficiency
- Distributed computing:

  masters Uncertainty

(We are -more or less- implicitly using a lot of heuristics!)

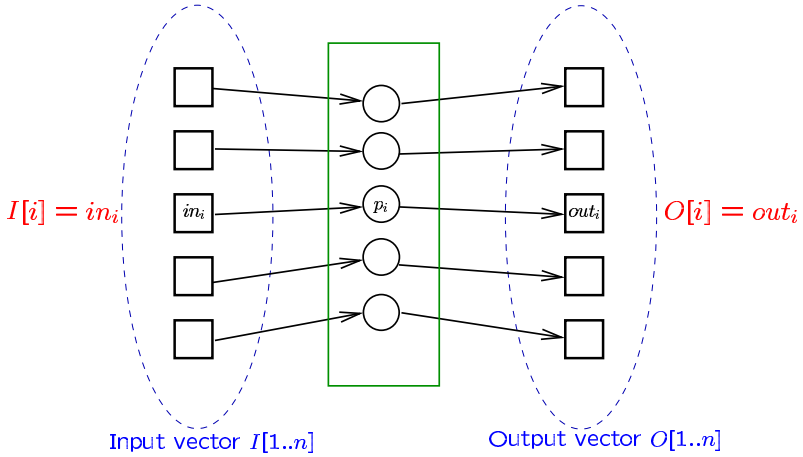
- Raynal M., Communication and Agreement Abstractions for Fault-Tolerant Asynchronous Distributed Systems. *Morgan & Claypool Publishers*, 251 pages, 2010 (ISBN 978-1-60845-293-4)

Uncertainty is created by

- Multiple loci of control
- Asynchrony (vs Synchrony)
- Failures (Failure models)
- Process mobility (and related stuff)
- Low computing capacity, bandwidth
- Dynamicity
- Etc., etc., ..., etc.!

The notion of a task in DC

The DC counterpart of a function



The notion of a task in DC

- A decision task T is a triple $(\mathcal{I}, \mathcal{O}, \Delta)$
  - $\star$ $\mathcal{I}$: set of input vectors (of size n)
  - $\star$ $\mathcal{O}$: set of output vectors (of size n)
  - $\star$ Δ: relation from $\mathcal{I}$ into $\mathcal{O}$: $\forall I \in \mathcal{I}: \Delta(I) \subseteq \mathcal{O}$
- $I[i]$: private input of p_i
- $O[i]$: private output of p_i
- $\forall I \in \mathcal{I}$:
  $\Delta(I)$ defines the set of output vectors that can be decided from the input vector I

Solving a task

A distributed algorithm A is a set of n local automata (Turing machines) that cooperate through specific communication objects (e.g., message-passing network, shared memory, etc.)

The set of automata is fixed (not a dynamic system with churn, etc.)

An algorithm A solves a task T if in any run

- $\forall\, I \in \mathcal{I}$ such that each p_i starts with (proposes) $in_i = I[i]$
- $\exists\, O \in \Delta(I)$ such that $out_j = O[j]$ for each process p_j that that computes (decides) an output out_j

Examples of tasks

- Consensus and k-set agreement
  - $\star$ Binary consensus:
    $\mathcal{I}$ = {all vectors of 0 and 1}
    $\mathcal{O}$ = $\{\{0,\ldots,0\},\{1,\ldots,1\}\}$
    Let $X_0 = \{0,\ldots,0\}$ and $X_0 = \{1,\ldots,1\}$
    Δ(any vector but $X_O, X_1) = \mathcal{O}$
    $\Delta(X_0) = \{0,\ldots,0\}$ and $\Delta(X_1) = \{1,\ldots,1\}$.
- Renaming
- Weak symmetry breaking
- k-Simultaneous consensus
- Etc.

Type of a task

- Colorless: if a value is decided by a process can be decided/output by any other process in the same run
  - ⋆ Example: consensus, k-set agreement
- Colored: symmetry breaking tasks
  - ⋆ Examples: Renaming problem, weak symmetry breaking

Part III

Base read/write distributed model

Processes, atomic registers and wait-freedom

- n sequential asynchronous processes: p_1, ..., p_n
- Communicate through atomic read/write registers
- Up to $n-1$ processes may crash in a run (wait-free)
  - ⋆ If a process crashes in a run it is faulty in that run, otherwise it is correct in that run

Wait-freedom

- An object implementation is *wait-free* if any invocation of any of its operations always terminates (i.e., whatever the behavior of the other processes) when the invoking process is correct

- Herlihy M.P., Wait-Free Synchronization. *ACM Transactions on Programming Languages and Systems*, 13(1):124-149, 1991

The fundamental issue

- Leslie Lamport: A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable!
- Impossibility of designing a deterministic algorithm solving consensus in an asynchronous distributed system
  - ⋆ Whatever the communication medium (read/write registers or message-passing)
  - ⋆ Even if a single process may crash
  - ⋆ Even if processes have to agree on a single bit!

- Fischer M.J., Lynch N.A., and Paterson M.S., Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374-382, 1985

- Raynal M., *Communication and agreement abstractions for fault-tolerant asynchronous distributed systems*. Morgan & Claypool, 251 pages, 2010 (ISBN 9781608452934)

Part IV

Distributed recursive algorithms

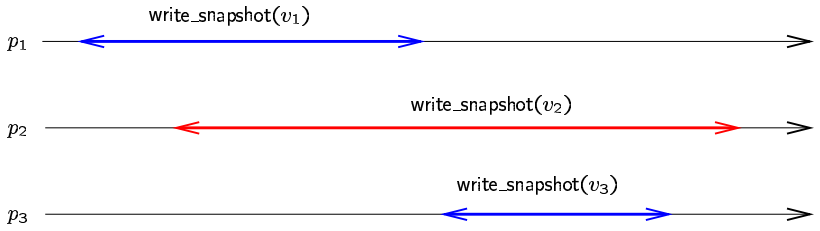
One-shot immediate snapshot object (1)

- A single operation write_snapshot()
- Allows the invoking process p_i to deposit a pair $\{\langle i, v_i\rangle\}$ and returns a set (called $view_i$) of pairs $\{\langle j, v_j\rangle\}$ where $\{\langle j, v_j\rangle\}$ has been deposited by p_j
- Specification
  - ⋆ Self-inclusion. $(i, v_i) \in view_i$.
  - ⋆ Containment. $\forall i, j$: $view_i \subseteq view_j$ or $view_j \subseteq view_i$.
  - ⋆ Immediacy. $\forall i, j$: if $(j, v_j) \in view_i$ then $view_j \subseteq view_i$.
  - ⋆ WF termination. The invocation of write_snapshot() by a correct process terminates.

- Borowsky E. and Gafni E., Immediate atomic snapshots and fast renaming. *12th ACM Symp. on Principles of Distr. Computing (PODC'93)*, pp. 41-51, 1993

One-shot immediate snapshot object (2)

An execution of an immediate snapshot object



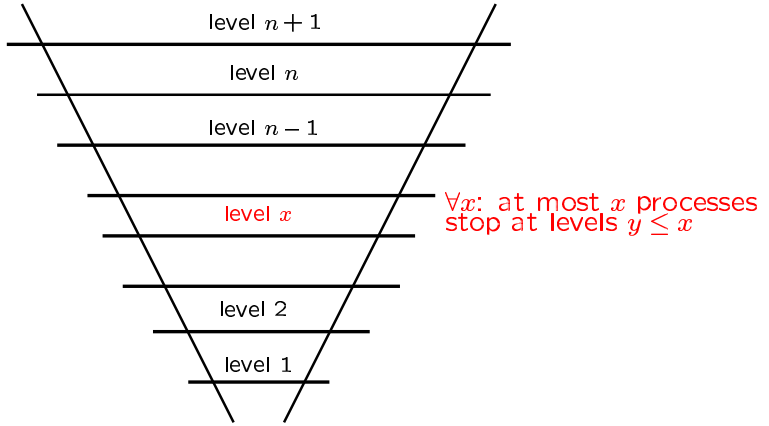
This concurrent execution is non-deterministc: 5 possible results

Iterative immediate snapshot from R/W registers

init: $\forall j:\ LEVEL[j] = n+1$.

operation write_snapshot(v_i) **is**
  $REG[j] \leftarrow v_i$;
  repeat $LEVEL[i] \leftarrow LEVEL[i] - 1$;
    for each $j \in \{1,\ldots,n\}$ **do** $level_i[j] \leftarrow LEVEL[j]$;
    $set_i \leftarrow \{x \mid level_i[x] \leq level_i[i]\}$
  until $(|set_i| \geq level_i[i])$ **end repeat**;
  let $view_i = \{\ (x, REG[x]) \mid x \in set_i\ \}$;
  return($view_i$)
end operation.

How it works



If p_i stops at level x, its view includes all the pairs $\langle j, v_j \rangle$ such that p_j has stopped at a level $y \leq x$

Recursive immediate snapshot from R/W registers (1)

operation write_snapshot(v_i) **is**
  $my\_view_i \leftarrow$ rec_write_snapshot(n, v_i)
  return(my_view_i)
end operation.

- Recursion parameter: n (nb of processes)

- Gafni E. and Rajsbaum S., Recursion in distributed computing. *Proc. 12th Int'l Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS'10)*, Springer-Verlag, #LNCS 6366, pp. 362-376, 2010

Recursive immediate snapshot from R/W registers (2)

init (recursion level x): $REG[x] \leftarrow [\bot,\ldots,\bot]$.

operation rec_write_snapshot(x, v) **is**
  % x is the recursion parameter $(n \geq x \geq 1)$ %
  $REG[x][i] \leftarrow v$;
  for each $j \in \{1,\ldots,n\}$ **do** $reg_i[j] \leftarrow REG[x][j]$ **end for**;
  $view_i \leftarrow \{\ (j, reg_i[j]) \mid reg_i[j] \neq \bot\ \}$;
  if $|view_i| = x$ **then** $res_i \leftarrow view_i$
    else $res_i \leftarrow$ rec_write_snapshot$(x-1, v)$
  end if;
  return(res_i)
end operation.

Distributed recursion: Linear time (1)

p_1	p_2	p_3	p_4	p_5
		rws$(5, v_3)$		
		rws$(4, v_3)$		
		crashes*		
			rws$(5, v_4)$	
			... rws$(1, v_4)$	
			$\{(4, v_4)\}$	
rws$(5, v_1)$	rws$(5, v_2)$			
rws$(4, v_1)$	rws$(4, v_2)$			
$view_1$	$view_2$			

where $view_1 = view_2 = \{(1, v_1), (2, v_2), (3, v_3), (4, v_4)\}$

* after it has written into REG

Distributed recursion: Linear time (2)

And, if p_3 has not crashed (it is only very slow)

p_1	p_2	p_3	p_4	p_5
		rws$(5, v_3)$		
		rws$(4, v_3)$		
		stops*	rws$(5, v_4)$	
			... rws$(1, v_4)$	
			$\{(4, v_4)\}$	
rws$(5, v_1)$	rws$(5, v_2)$			
rws$(4, v_1)$	rws$(4, v_2)$			
$view_1$	$view_2$	wakes up		
		rws$(3, v_3)$		
		rws$(2, v_3)$		
		$\{(3, v_3), (4, v_4)\}$		

Gain

- Easier proof
- Better step complexity (shared memory accesses): $O(n(n - |view| + 1)$ instead of $O(n^2)$
- Better understanding: capture the essence of the problem, namely, what has to be captured is the *concurrency pattern/(a)synchrony*

Adaptive renaming (1)

- Each process p_i has an initial name $id_i \in [1..N]$
- A single operation new_name()
- Allows the invoking processes to obtain new names in a name space $[1..M]$ with $M << N$
- Lower bound $M = 2n - 1$ ($2n - 2$ for some values on n)
- One-shot vs long -lived renaming (Wait-freedom)

- Attiya H., Bar-Noy A., Dolev D., Peleg D. and Reischuk R., Renaming in an asynchronous environment. *Journal of the ACM*, 37(3):524-548, 1990.

- Herlihy M.P. and Shavit N., The topological structure of asynchronous computability. *JACM*, 46(6):858-923, 1999.

- Castañeda A. and Rajsbaum S., New combinatorial topology upper and lower bounds for renaming: the lower bound. *Distrib. Computing*, 22(5):287-301, 2010

Adaptive renaming: specification

Let p be the nb of participating processes ($1 \leq p \leq n$)

- Validity. A new name belongs to the set $[1..2p-1]$.
- Agreement. No two processes obtain the same new name.
- Index independence. $\forall\, i, j$, if a process whose index is i obtains the new name v, that process could have obtained the very same new name v if its index had been j.

  $\Rightarrow$ indexes can be used only for addressing purposes
- WF termination. The invocation of new_name() by a correct process does terminate

Adaptive renaming: iterative algorithm

- A conceptually simple iterative algorithm is described in

  Attiya H and Jennifer W., Distributed computing: fundamentals, simulations and advanced topics, (2d Edition), *Wiley-Interscience*, 414 pages, 2004 (ISBN 0-471-45324-2)
- SM version of an MP algo. by Attiya et al. (JACM'90)
- It is based on a snapshot object
- Its step complexity can be exponential
- It is optimal wrt the size of the new name space

Adaptive renaming: two recursive algorithms

- Borowsky E. and Gafni E.,
  Immediate atomic snapshots and fast renaming.
  Proc. 12th ACM Symposium on Principles of Distributed Computing (PODC'93), pp. 41-51, 1993
- Rasjbaum S. and Raynal M.,
  A theory-oriented introduction to wait-free synchronization based on the adaptive renaming problem.
  Proc. 25th Int'l Conference on Advanced Information Networking and Applications (AINA'11), IEEE Press, pp. 356-363, 2011.

Adaptive renaming: a simple idea (Gafni)

- Let $n = 2$ processes
- When, it wants to acquire a new name a process first writes its initial name into the shared memory to indicate it participate
- Then, two cases:
  - ⋆ If it sees only itself, it takes new name 1
  - ⋆ Otherwise
    - ∗ if its initial name is greatest than the one of the other process it takes new name 2,
    - ∗ if its initial name is smaller it takes new name 3
- New name sapce is $[1..2p-1]$ ($p = \#$ part. processes)

Adaptive renaming: RR'10 algorithm (1)

- Notation: $up = 1 = \overline{down}$ and $down = -1 = \overline{up}$
- A process first invokes new_name$(n, 1, up)$ indicating it wants a new name in the set $[1..2n-1]$
- More generally, a process invokes new_name$(x, first, dir)$ indicating that it wants a new name in a set defined from x, $first$ and $dir \in \{up, down\}$
- Each process
  - ⋆ Progresses from the recursion level n to the recursion level x at which it sees x participating processes
  - ⋆ Then competes for new names only from that level
  - ⋆ In an interval of size $2x-1$
    - ∗ $[first..first + (2x-2)]$ id $dir = up$
    - ∗ $[first - (2x-2)..first]$ id $dir = down$

Adaptive renaming: RR'10 algorithm (2)

operation new_name$(x, first, dir)$ **is**
  % x $(n \geq x \geq 1)$ is the recursion parameter %
  $SM[x, first, dir]$.store(id_i);
  $competing_i \leftarrow SM[x, first, dir]$.collect();
  if $|competing_i| = x$
    then CONFLICT RESOLUTION
      SOLVED FROM $SM[x, first, dir]$
    else $res_i \leftarrow$ new_name$(x-1, first, dir)$
  end if;
  return(res_i)
end operation.

Adaptive renaming: RR'10 algorithm (3)

An invocation of new_name$(x, first, dir)$

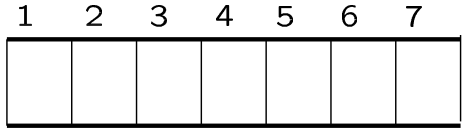
- Init: $SM[x, first, dir] \leftarrow [\perp, \ldots, \perp]$.
- $\leq x$ processes access $SM[x, first, dir]$ and
  - ⋆ $\leq x-1$processes invoke new_name$(x-1, first, up)$ and will rename in $[first..first + 2(x-2)]$
  - ⋆ $\leq x-1$ processes invoke new_name$(x-1, last + \overline{dir}, \overline{dir})$ where $last = first + dir(2x-2)$
  - ⋆ At most 1 process stops
- : Splitter-like object!

Adaptive renaming: RR'10 algorithm (4)

operation new_name$(x, first, dir)$ **is**
  % x $(n \geq x \geq 1)$ is the recursion parameter %
  $SM[x, first, dir]$.store(id_i);
  $competing_i \leftarrow SM[x, first, dir]$.collect();
  if $|competing_i| = x$
    then * $last \leftarrow first + dir(2x-2)$;
      * **if** $id_i = \max(competing_i)$
      * **then** $res_i \leftarrow last$
      * **else** $res_i \leftarrow$ new_name$(x-1, last + \overline{dir}, \overline{dir})$
      * **end if**
    else $res_i \leftarrow$ new_name$(x-1, first, dir)$
  end if;
  return(res_i)
end operation.

Distributed recursion: Branching space/time (1)

- $n = 4$ processes p_1, ..., p_4
- Size of the adaptive renaming space $m = 2p - 1$



Distributed recursion: Branching space/time (2)

$SM[4, 1, up]$

p_3 invokes new_name$(4, 1, up)$

p_3 invokes new_name$(3, 1, up)$

p_3 invokes new_name$(2, 1, up)$

p_3 invokes new_name$(1, 1, up)$
p_3 obtains the new name 1

As p_3 has seen $p = 1$ process (itself)
it decides the new name $2p - 1 = 1$

After p_3 has obtained the new name 1
p_1 and p_4 invoke concurrently new_name$(4, 1, up)$
which entail their concurrent invocations of new_name$(3, 1, up)$

Distributed recursion: Branching space/time (3)

Let $id_1, id_4 < id_3$
p_1 and p_4 invoke new_name$(3, 1, up)$, they see $p = 3$ processes and both compute $last = 1 + (2 * 3 - 2) = 5 \Rightarrow$ their new name space = [1..5]

First p_4 executes alone and
invokes new_name$(2, 4, down)$

Then, p_4 invokes new_name$(1, 4, down)$
$last = 4 - (2 * 1 - 2) = 4$ and p_4 decides 4

Later p_1 invokes new_name$(2, 4, down)$
If $id_4 < id_1$; p_1 decides $last = 4 - (2 * 2 - 2) = 2$
If $id_1 < id_4$: p_1 invokees new_name$(1, 3, 1)$ and decides 3

Later p_2 invokes new_name$(4, 1, up)$ and sees $p = 4$ processes
p_4 computes $last = 1 + (2 * 4 - 2) = 7$ and invokes
new_name$(3, 6, -1)$, new_name$(2, 6, -1)$, new_name$(1, 6, -1)$
p_2 then computes $last = 6 + (2 * 1 - 2) = 6$ and decides 6

Part V

Conclusion

Recursion and fault-tolerant distributed computing

- Important research area for distributed tasks
- Wait-free implementation (strongest fault-tolerance wrt liveness)
- Recursion parameter: the nb of processes
- Algorithms have to capture the concurrency degree seen by a process in order to allow it to solve conflicts in an appropriate way
- Far from and much involved than data independence as exploited in (recursive) parallelism
- What about the relations linking Data structures, Control structures and Fault-tolerance in DC?

Short bibliography on Distributed recursion

- Castañeda A., Rajsbaum S. and Raynal M., The renaming problem in shared memory systems: an introduction. *Elsevier Computer Science Review*, 5:229-251, 2011
- Gafni E. and Rajsbaum S., Recursion in distributed computing. *Proc. 12th Int'l Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS'10)*, Springer LNCS 6366, pp. 362-376, 2010
- Raynal M., Chapters 8 & 9 in *Concurrent programming: algorithms, principles and foundations principles*, Springer, 515 pages, 2013 (ISBN 978-3-642-32026-2).

The only slide to remember

Failures do modify

- Our view of synchronization
- The way synchronization has to be solved