Distributed Computing

Pierre Fraigniaud

INSTITUT DE RECHERCHE EN INFORMATIQUE FONDAMENTALE

CNRS and Université de Paris

20th Max Planck Advanced Course on the Foundations of Computer Science (ADFOCS)
19 - 23 August 2019, Saarbrücken, Germany
Algorithms vs. programs

Mechanical procedures for solving a given problem
Distributed Algorithm

A collection of autonomous computing entities collaborating for solving a task in absence of any coordinator
Parallel vs. Distributed

Parallel computing

Performances
> petaFLOPS ($10^{15}$ op./s)

Distributed computing

Coping with uncertainty
temporal and spatial
Sequential vs. Distributed

Alan Turing

Alonzo Church

Typical model for distributed computing

Communication Medium
Communication Medium

Processing elements (PE)

Shared memory

Network

Memory
Limitations Faced by Distributed Computing: Undecidability + Uncertainty

Sources of uncertainties:

• Spatial: communication network
• Temporal: clock drifts (asynchrony, load, etc.)
• Failures (transient, crash, malicious, etc.)
• Selfish behavior (game theory)
• ...

Several Turing machines are weaker than one!
Symmetry Breaking

- Leader election
- Consensus
- Coloring
- Graph problems
- Etc.

Applications:

- Frequency assignments
- Distributed data-bases consistency
ADFOCS Lectures

- Asynchronous Crash-Prone Distributed Computing
- Locality in Distributed Network Computing
- Congestion-Prone Distributed Network Computing
- Other Aspects of Distributed Computing

¹ See also lecture by Cristoph Lenzen on Wednesday
Reference books
ADFOCS Lectures

- Asynchronous Crash-Prone Distributed Computing
- Locality in Distributed Network Computing
- Congestion-Prone Distributed Network Computing
- Other Aspects of Distributed Computing
Temporal Uncertainty

Dealing with *asynchronism*:
- clock drifts
- cache misses
- poor load balancing
- etc.

and *failures*:
- crash failures
- transient failures
- byzantine (i.e., adversarial) failures
- etc.
Computing Model

Processing elements, a.k.a. processes

- write(value)
- read(register index)

My ID is 1010
My input is 1

Shared memory
Consensus

Termination: every correct process decides a value 0 or 1.

Agreement: all the decided values are identical.

Validity: every decided value must have been proposed.

Distributed replica, mutual exclusion, etc
Impossibility of Consensus

M. Fischer, N. Lynch, M. Paterson (1985)

**Theorem** Binary consensus cannot be solved in a shared-memory asynchronous system, even with at most one crash failure.

Dijkstra Prize 2001
Proof
(in the case of any #failures)

• Also known as the **wait-free** model

• **Extension-based proof:** sequence of system configurations for which no processes can decide
  \[ C^{(0)}, C^{(1)}, C^{(2)}, C^{(3)}, \ldots \]

• **Time = Scheduler**

• **Bivalent** vs. **monovalent** configurations

• Monovalent configuration: 0-valent or 1-valent
Claim 1 There exists an initial bivalent configuration

Proof. Assume all init configurations are monovalent. 
$C_0 = 00\ldots0$ is 0-valent  
$C_n = 11\ldots1$ is 1-valent  
Let $k$ be smallest index such that $C_k$ is 1-valent 
\[
\begin{array}{c}
11\ldots100\ldots00 \\
11\ldots110\ldots00 \\
\end{array}
\]
Scheduler crashes process $p_k$ 
⇒ other processes cannot distinguish $C_{k-1}$ from $C_k$  
Contradiction!

❑
C ~_p C’ if C and C’ looks the same from process p

**Claim 2** Let C and C’ be two monovalent configurations. If C ~_p C’ then C and C’ have the same valency.

*Proof.* The scheduler crashes all processes but p.

»
A process \( p \) is critical for a bivalent configuration \( C \) if \( p \) taking a step in \( C \) results in a monovalent configuration.

**Claim 3** For every bivalent configuration \( C \), there exists a process \( p \) that is **not** critical for \( C \).

**Proof.** Assume every process is critical.

\[
p \to 0\text{-valent} \quad \text{and} \quad q \to 1\text{-valent}
\]

- Case 1: \( p \) and \( q \) both read, or they read or write in different registers
  \[
  \Rightarrow Cpq = Cqp, \text{ contradiction.}
  \]

- Case 2: \( p \) reads or writes in \( R \), and \( q \) writes in \( R \)
  \[
  \Rightarrow Cq \sim_q Cpq, \text{ contradiction with Claim 2.}
  \]

\[\Box\]
Weak Consensus

- **Termination**: every correct process decides a value 0 or 1, or ⊥ (i.e., aborts).
- **Agreement**: all the decided values ≠ ⊥ are identical.
- **Validity**: If no processes crash, then at least one process must decide a proposed value.
**Property** Weak consensus is solvable wait-free in asynchronous shared-memory systems.

The algorithm uses *snapshot* instructions

snapshot = atomic read of the entire memory (i.e., all the registers)

**Lemma** Atomic snapshot can be implemented wait-free.

**Remark** *Immediate* snapshot — write-snapshot as a single atomic operation — can also be implemented wait-free.
Algorithm of process $p$ with input value $v$

begin
write ($p,v$)
snapshot
let $V = ((p_1,v_{p_1}),\ldots,(p_k,v_{p_k}))$ /*the view of $p$*/
write ($p,V$)
snapshot
let $W = ((p_1,V_{p_1}),\ldots,(p_m,V_{p_m}))$ /*the meta-view of $p$*/
let $V^* = \bigcap_{i=1,\ldots,m} V_{p_i}$ /*smallest view in the meta-view of $p$*/
if for every $i \in [1,n]$ such that $v_i \in V^*$, $V_i \in W$ holds
then decide smallest value in $V^*$
else decide $\perp$
end
Intuition

my value v

my view V

my meta-view W
Termination trivially holds

**Claim 1** Agreement holds

*Proof* Assume $p$ decides $v \neq \bot$, and $p'$ decides $v' \neq \bot$ with $v < v'$.

Let $q \neq q'$ such that $V^*_p = V_q$ and $V^*_p' = V_{q'}$.

- On the one hand: $v \notin V_{q'}$ since $p'$ decides $v' > v$.

Therefore $V_{q'} \subset V_q$, and thus $v_{q'} \in V_{q} = V^*_p$

- On the other hand: $V_{q'} \notin W_p$ as otherwise $V^*_p = V_{q'}$

Contradiction: $p$ does not satisfy the if-condition.
Claim 2 Validity holds

Proof

• If \( p \) decides \( \perp \) then there exists \( q \neq p \) such that \( q \) performed its first write before the first snapshot of \( p \), and \( p \) performed its second snapshot before the second write of \( q \).

\[
\text{write}_1 \quad \text{snap}_1 \quad \text{snap}_2 \quad \text{write}_2
\]

• Assume all proc decide \( \perp \).

\[
\text{snap}_2 \quad \text{write}_2 \quad \text{snap}_2 \quad \text{write}_2 \quad \text{snap}_2 \quad \text{write}_2 \quad \text{snap}_2
\]
Combinatorial Topology
Configurations

System configurations at time $t$

(global state) $S \rightarrow (p,v) \rightarrow (q,w) \rightarrow S'$

(process) $S \rightarrow (p,v) \rightarrow S'$

(state/view) $S \rightarrow (q,w) \rightarrow S'$
Simplexes and Complexes

- A complex is defined as a pair $K = (\mathcal{V}, \mathcal{S})$ where
  - $\mathcal{V}$ is the (finite) set of vertices
  - $\mathcal{S}$ is a collection of non-empty subsets of $\mathcal{V}$, closed under vertex deletion, i.e., $S \in \mathcal{S} \implies \forall S' \subseteq S, S' \in \mathcal{S}$.
  
Every $S \in \mathcal{S}$ is a simplex.

- Examples:
  - $G=(\mathcal{V},E)$ defines the complex $K=(\mathcal{V}, E \cup \mathcal{V})$
  - A higher dimensional complex:
The configurations of a distributed system at time $t$ defines the **protocol complex** $P_t = (V, S)$ with

- $V = \{(p,v), p \text{ process, } v \text{ state of } p \text{ at time } t\}$
- $S \in \mathcal{P}(V)$ belongs to $S$ if $S$ is a set of views from different processes, corresponding to a same execution $\mathcal{E}$
Input/Output Complexes and Task Specification

Input binary consensus

Output binary consensus
Task Solvability

Theorem Task \((I,O,\Delta)\) is solvable iff there exists \(\delta\) such that, for every \(S \in I\),
\[\delta(\xi(S)) \in \Delta(S)\]
Wait-Free Computing

Asynchronous

Shared Memory

CPU CPU CPU CPU

Interconnect

Memory

Crash failures

Alice hasn’t seen Bob
Bob saw Alice

Alice saw Bob
Bob saw Alice

Three Processes
(iterated immediate snapshot)

All possible executions during one step.

A facet of the simplex

Several facets together
Wait-Free Solvability

M. Herlihy and N. Shavit (1999)

**Theorem** A task is solvable in the asynchronous model with crashes if and only if there exists a simplicial map from a *chromatic subdivision* of the input complex to the output complex, respecting the specification of the task.

Gödel Prize 2004
No simplicial map from a subdivision of the input complex to the output complex respecting the specifications of consensus.
Figure 17: Subdivision du complexe d’entrée du consensus binaire après deux étapes.

Processeurs. En revanche, pour tout nombre de processeurs, le complexe de sortie du consensus binaire n’est pas connexe. On observe alors les faits suivants :

– D’une part, l’image par une application simpliciale d’un complexe connexe étant connexe, toute application simpliciale


d’une subdivision \( K \) de \( I \) vers \( O \) doit satisfaire

\( (K) = C \), où \( C \) est l’une ou l’autre des deux composantes connexes de \( O \).

– D’autre part, doit être compatible avec \( \bigcirc \), ce qui implique en particulier que l’image


de tout simplexe issu d’une subdivision du simplexe

\( \{ (p_1,0),\ldots,(p_n,0) \} \) de \( I \) doit avoir pour image le simplexe

\( \{ (p_1,0),\ldots,(p_n,0) \} \) de \( O \), et tout simplexe issu d’une subdivision du simplexe

\( \{ (p_1,1),\ldots,(p_n,1) \} \) de \( I \) doit avoir pour image le simplexe

\( \{ (p_1,1),\ldots,(p_n,1) \} \) de \( O \).

Les deux faits ci-dessus sont en contradiction, et l’application


ne peut exister, d’où il découle

par le théorème 14 qu’il est impossible de résoudre le consensus dans un modèle asynchrone sans attente. La cause de cette impossibilité est topologique : résoudre le consensus demande de transformer par subdivision un complexe connexe en un complexe non connexe, ce qui n’est pas possible.

En guise de conclusion à cette brève introduction à l’utilisation de la topologie algébrique pour l’étude de la calculabilité en distribué, observons que la connectivité n’est pas un obstacle dans le cas du consensus faible, pour lequel les processeurs ont l’option de retourner \( \bot \) en valeur de sortie. En effet, comme illustré sur la figure 18, cette option “reconnecte” le complexe de sortie.
Variants

k-set agreement
• n processes with input values in \{1,\ldots,m\}
• objective: agree on at most k proposed values
• remark: (n-1)-set agreement is called set-agreement

t-resilient model
• asynchronous
• t = maximum number of crash failures
Set-agreement solvability

**Theorem** In the t-resilient model, if $k \geq t+1$, then $k$-set agreement is solvable.

*Proof* Algorithm for $(t+1)$-set agreement in the t-resilient model:

```
begin
  repeat
    snapshot
  until values from at least $n-t$ processes are seen
  decide minimum seen value
end
```

➤ at most $t+1$ different views
Topological perspective

\( t = 1 \)

yields holes in the protocol complex

enables to map the protocol complex to the output complex
Other applications of topology to distributed computing

• Processes occupy nodes of a graph G
• Synchronous model
• Communication by messages
• No failures
• Graph G is known to every process, including the position of every other process.
A dominating set in $G=(V,E)$ is a set $D \subseteq V$ such that every node not in $D$ has a neighbor in $D$.

**Definition** $G$ has dominating number $d$ if the min size of a dominating set in $G$ has cardinality $= d$.

**Theorem** $k$-set agreement in $G$ requires at least $r$ rounds where $r$ is the minimum integer such that $G^r$ has dominating number $\leq k$. 
Proof for $m=3$ and $k=2$

Input configuration: $v_1 v_2 \ldots v_n$ with $v_i \in \{0, 1, 2\}$

For every $i,j$, there exists process $q$ that is not dominated by $p_i$ nor $p_j$.

These triangles can be glued together.
Assume existence of an algorithm.

Colored each node by the discarded color

Remark:

Impossible

The coloring of the border nodes is forced
Sperner’s Lemma

**Lemma** Every Sperner coloring of a triangulation of an $n$-dimensional simplex contains a cell colored with a complete set of colors.
Proof sketch

- By induction on n: $\deg(u)$ is odd
- $\sum_{v \in V(G)} \deg(v) = 2 |E(G)|$
- Triangles with 1 or 2 colors induce nodes with even degrees (0 or 2)

$\Downarrow$

Odd number of 3-colored triangles
Concluding remarks
Message Passing vs. Shared Memory

- Processing elements (PE)

- Shared memory

- Message passing
Equivalence

H. Attiya, A. Bar-Noy, D. Dolev (1990)

**Theorem** The message-passing and shared-memory models with crash failures are “essentially” equivalent

Dijkstra Prize 2011
Overcoming impossibility results

• **Failure detectors**: e.g., T. Chandra, V. Hadzilacos, S. Toueg (1995)

• **Randomization**: e.g., Ben-Or Algorithm for consensus (1983)

• **Best-effort algorithms**: e.g., *Paxos* algorithm (1989) by L. Lamport (Turing Award 2014)

• **Build-in atomic objects**: beyond read/write registers, like *test&set, compare&swap*, etc.
Open problems

• **Renaming:**
  ‣ $n$ processes start with unique names taken from a large name space $[1,N]$
  ‣ they must decide new unique names from a name space as small as possible.
  ‣ Result: $2n-1$ possible; optimal for infinitely many $n$, but not for all $n$.

• **Algebraic topology:**
  ‣ Randomized algorithms
  ‣ Byzantine failures

• **Distributed verification**
  ‣ Proving correctness using formal methods and/or proof assistants
  ‣ Distributed monitoring