Distributed computing on mobile tiny devices

Rachid Guerraoui
EPFL
A world of tiny mobile devices
What computing model?
Overview

(1) Mobility
(2) Crashes
(3) Privacy
(4) Security
The march of the penguins
The group is threatened if more than a threshold dies on the way back to the sea.

If a penguin starts its trip with a very low temperature, the probability that it reaches the sea is very low.
The escort

Provide each penguin with a computing device to:

- measure its temperature;
- trigger an alert if a threshold (say 5) has a very low temperature
Assumptions

- Every device holds a finite counter (<6)
  - Its initial value is 1 if the penguin has a low temperature and 0 otherwise

- A pair of devices communicate if they get close enough
  - Every pair of devices eventually meet
The problem

All devices eventually output “alert” iff at least 5 initial values are 1
Algorithm

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>1</th>
<th></th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
</tr>
</tbody>
</table>

The diagram shows a sequence of operations with arrows connecting the steps, indicating the flow of the algorithm.
Algorithm?
Algorithm

When two devices meet, one keeps in its counter the sum of the values whereas the other puts it back to 0.

Any device with value 5 triggers the alert.
<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Algorithm**
### Algorithm

```
0 0 0 0 0 0 0 1 1 1 1

0 0 0 0 0 0 0 0 0 2 1

0 0 0 0 0 0 0 0 0 3 0

0 0 0 0 0 0 0 0 0 3 0
```
Population protocols (DF01, AADFP’04)

A population $\mathbb{P}$ is a set of agents.

Every agent has a bounded memory independent of the size of the system.

Algorithms are uniform and anonymous.
Population protocols

The agents are asynchronous and have no control over their mobility pattern.

A pair of agents communicate if they get close to each other (one is the initiator).

Every interaction that is always possible eventually happens (fairness).
Population protocol

Input; Output; State S
InMap In; OutMap Out
Transition $\delta$
Configuration / Execution

A *configuration* is a set of states of all agents.

An *execution* is a sequence of configurations.
Fairness

An infinite execution $E$ is *fair* if for any transition $\delta(C,C')$, if $C$ appears infinitely in $E$, then $C'$ appears infinitely in $E$.

A *computation* is a fair execution.
**Stability and convergence**

A configuration $C$ is **stable** if for every $C'$ such as $\delta(C,C')$, we have: $\text{Out}(C) = \text{Out}(C')$

An infinite execution **converges** if it has a stable configuration
Back to the Penguins

- Input = \{0,1\}
- Output = \{ALARM, OK\}
- S = \{0,1,2,3,4,5\}
- In (identity): 0 \rightarrow 0, 1 \rightarrow 1
- Out: \{0, 1, 2, 3, 4\} \rightarrow OK
  5 \rightarrow ALARM
Transition

$\delta: (i, j) \rightarrow (i+j, 0) \text{ if } i+j < 5$

$(5, 5) \text{ if } i+j \geq 5$
More generally (AAD06)

Theorem: population protocols compute exactly first order Presburger’s arithmetic: +, -, =, >, or, not, and, ...

NB. Not as powerful as Peano’s arithmetic which also include *
Overview

(1) Mobility
(2) Crashes
(3) Privacy
(4) Security
But what if?

One of the agents fail (say by crashing at some inappropriate time)

An agent might crash exactly when it reaches value 4
Original algorithm O
**Reliable algorithm**

Every agent performs *twice* the original algorithm: O1 and O2

When two agent communicate, one acts as the initiator for O1 and the other as the initiator for O2
Reliable algorithm
More generally (DFGR06)

Theorem: population protocols compute exactly Presburger’s arithmetic with a constant number of crashes
This talk

(1) Mobility
(2) Crashes
(3) Privacy
(4) Security
What about privacy?

How can we hide the initial values from curious agents?

How can we compute a result while preventing any agent from figuring out, at any point in time, any information besides its own input and the result of the computation?
What about privacy?

How can we make it impossible for a curious agent to distinguish the situation where exactly 5 penguins have low temperature from the situation where more do?
Original algorithm
How to ensure privacy?

- An agent cannot use crypto (not even signatures because of anonymity)

- An agent can see the entire state of the agent it is interacting with: hence no secret keys are possible
Obfuscation
More generally (DFGR07)

Theorem: population protocols can privately compute exactly first order Presburger’s arithmetic
This talk

(1) Mobility
(2) Crashes
(3) Privacy
(4) Security
What if agents can be malicious?

What can we compute with one malicious agent? (arbitrary transitions)
Malicious agent
More generally (GR07)

Theorem: population protocols cannot compute any non-trivial predicate with a single malicious agent

NB. A predicate is *trivial* if every agent can determine its output based only on its input
Community protocols (GR07)

Every agent has:

- A bounded memory where it can execute arithmetics (as in the original population model)
- A bounded set of slots to store identities and test (only) for their equality
**Community protocols (GR07)**

*Theorem:* community protocols can exactly compute every *symmetric* predicate that can be (Turing) computed in \( n \log n \) space.