When Birds Die:
Making Population Protocols Fault-Tolerant

Carole Delporte-Gallet, LIAFA, Paris, France

Hugues Fauconnier, LIAFA, Paris, France

Rachid Guerraoui, EPFL, Switzerland & MIT, USA

Eric Ruppert, York University, Canada

Motivating Example: Birds

- Strap tiny, identical sensors to many birds in a flock.

- Sensors on two birds can interact when the birds are close together.

- Want to detect when (at least) five birds have elevated body temperatures, indicating possible epidemic.

[Angluin, Aspnes, Diamadi, Fischer, Peralta 2004]

Population Protocol Model

Agents are

- simple,
- identical,
- passively mobile.

Each agent has $O(1)$ memory space.

When two devices come near each other, they can interact and both can update their own states.

Fairness guarantees that all possible interactions eventually happen.

Original model assumed no failures.

[Angluin, Aspnes, Diamadi, Fischer, Peralta 2004]

Computing with a Population Protocol

- Each agent has an input value (from a finite set).

- Goal is to compute a function on the multiset of input values.

  Examples: Each agent has a binary input.
  - Are there at least 5 agents with input 1?
  - Is the sum of the inputs odd?
  - Majority (Are there more sick birds than healthy birds?)

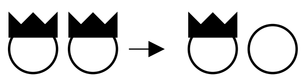
Every agent must eventually converge to the correct output.

(At any time, the state of an agent determines its current output. Output may change over time, but eventually stabilizes.)

Example Failure-Free Protocol: Majority

Problem: Are there more reds than blues among the inputs?

Each agent stores a colour, red, blue or purple,
and a leader bit (initially true).

Rule 1: 

$\Rightarrow$ Eventually only one leader left.

Rule 2: 

$\Rightarrow$ Eventually there are no reds or no blues.

Rule 3: 

$\Rightarrow$ Eventually, leader has majority colour (or purple in case of a tie).

Each agent remembers the colour of the last leader he met to determine his output.

Previous Work

With no failures, population protocols can:

- compute sum of inputs modulo a constant,
- compare linear combinations of inputs to a constant,
- compute boolean combinations of the above.

[Angluin, Aspnes, Diamadi, Fischer, Peralta 2004]

Generalization to restricted interaction graphs.

[Angluin, Aspnes, Chan, Fischer, Jiang, Peralta 2005]

One-way interactions. [Angluin, Aspnes, Eisenstat, Ruppert 2005]

Self-stabilizing population protocols that maintain certain properties.

E.g. leader election, token passing.

[Angluin, Aspnes, Fischer, Jiang 2005]

Failures

What if the birds drop dead (along with their sensors)?

Useful computations can be done in the presence of failures.

We consider two types of failures.

Crash failure: an agent crashes without warning.

Transient failure: an agent's state is corrupted.

Assume known bounds on number of failures to be tolerated:
c is maximum number of crash failures,
t is maximum number of transient failures.

Main Result

We give a general construction to transform any protocol for the failure-free model to a protocol that tolerates failures.

Making Problem Specification Fault-Tolerant

What if failures happen right away, obscuring some inputs?

Solution #1: Add preconditions on possible inputs.

Majority Example:
Must determine whether more inputs are red or blue.

Can tolerate c crash failures, if you know
$|\#\text{reds} - \#\text{blues}| > c$.
(Even if c agents crash, the winner is still clear.)

Can tolerate t transient failures, if you know
$|\#\text{reds} - \#\text{blues}| > 2t$.
(Even if t agents switch sides, the winner is still clear.)

Alternate Solution

Solution #2: Make the functions fuzzier.
If an input is close to an input that would produce output x,
make x an allowable output.

Threshold Example:
Must determine whether at least 5 birds have a fever.

With one halting failure,
you can solve a weaker version of the problem:
Output 1 if at least 5 birds have a fever,
Output 0 if at most 5 birds have a fever.

(Notice either output is allowed when exactly 5 birds are sick).

More Precisely ...

For this talk, we focus on Solution #1 (adding preconditions).

Let $\mathcal{D}$ be the domain of allowed inputs.
Let Y be the set of possible outputs.

The function $f : \mathcal{D} \rightarrow Y$ is computable if
it can be extended to all possible inputs such that
$\forall I \in \mathcal{D}$, removing up to $c+t$ values from I and adding up to t values
does not change the output value.

(Recall $c = \#$ crashes, $t = \#$ transient failures.)

The Main Idea

- Start with any protocol that does not tolerate failures.
- Use replication: Simulate many runs of the protocol.
- Ensure each failure interferes with at most two simulated runs.
- Use enough simulated runs that a majority will be correct.

How to Simulate Runs: Phase 1

Need $\Theta(n)$ space to simulate all agents.
$\Rightarrow$ each simulated run needs a constant fraction of agents to cooperate on the simulation.

Phase 1: Divide agents into g disjoint groups.
(g depends on c and t, but is constant with respect to n.)
Later, each group will collectively run one simulation.

- Split off one group at a time.
- Use division algorithm of Angluin *et al.*
- Each group has (roughly) n/g agents.

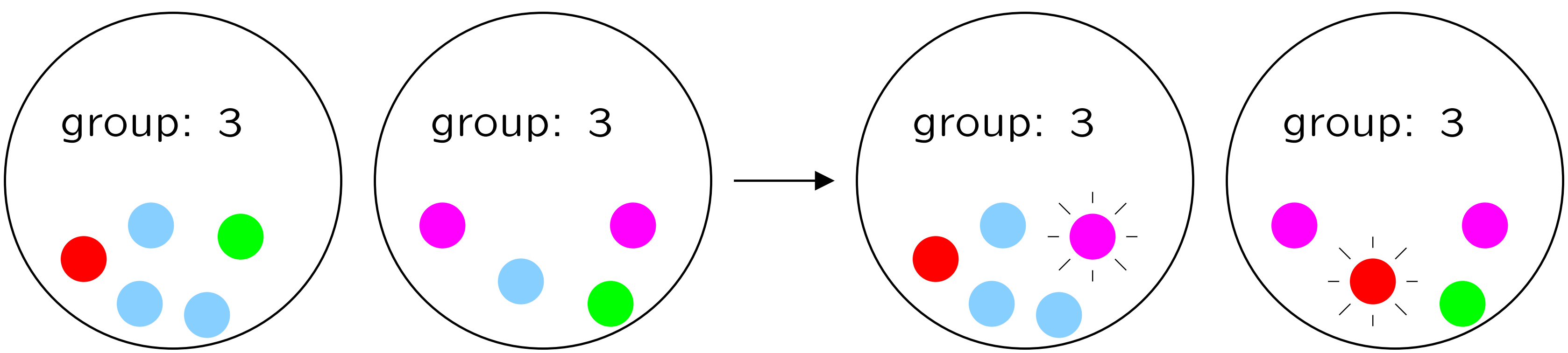
How to Simulate Runs: Phase 2

Phase 2: Each group simulates a run.

- Each agent in the group stores the states of approximately g simulated agents (called threads).

- An agent first gathers initial values from other agents to create its threads.

- When two agents in the same group meet, they simulate an interaction between two of their threads.



How to Simulate Runs: Phase 3

Phase 3: Producing the output:

- Remember the output of a thread of the last agent you saw from each group.
- Find the most common output among these values.

All threads in failure-free groups converge to correct output.

The number of groups (g) is chosen so that, eventually, a majority of the outputs you recorded are correct.

Difficulties

• Effect of failures must be contained.

Solution: A single agent should not have a critical role: no leaders. Also, carefully limit a failure's effects to one or two groups.

$\Rightarrow$ Sufficiently many groups run perfect simulations.

E.g. Failures of agents in phase 1 (splitting agents into groups).

Solution: Settle for approximate splitting.

• Agents do not know when a protocol (or part of it) is complete: hidden agents could suddenly appear, requiring more computation.

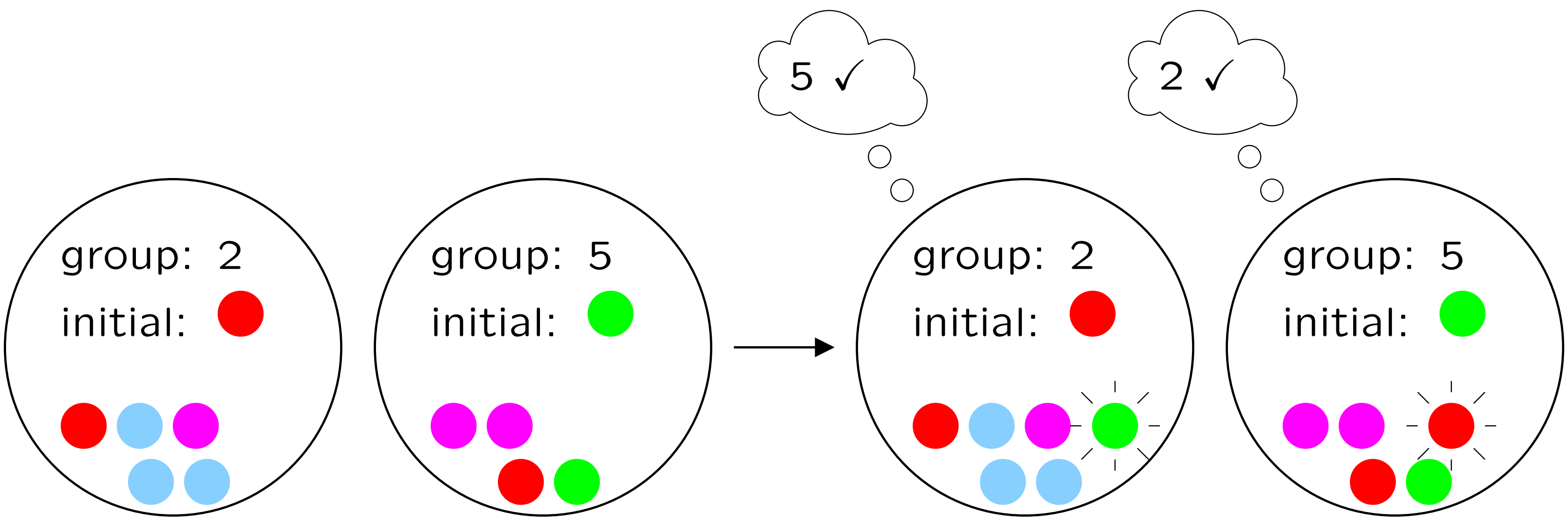
Solution: Allow phases to overlap.

Difficulties

- Anonymity and finite space per agent makes bookkeeping hard.

E.g. Must ensure each agent is simulated by exactly one thread in each group.

Solution: Create threads carefully.



$\Rightarrow$ Each transient failure creates at most one bogus input value in each group.

A Lower Bound

In the case of c crash failures only,
the sufficient condition becomes:

The function $f : \mathcal{D} \to Y$ is computable if
it can be extended to all possible inputs such that
$\forall I \in \mathcal{D}$, removing up to c values from I
does not change the output value.

The converse is also true:

We prove that any function f computable tolerating c crashes
satisfies this condition.

Future Directions

• We gave characterization of computable functions for crashes. There is still a gap for transient failures.

• What can be done when a constant fraction of agents fail?

• Recent work characterizes computable functions with no failures. [Angluin, Aspnes, Eisenstat PODC 06]
$\Rightarrow$ May allow simplified transformation.

• Study complexity in population protocol model.

Thanks.