I. The Prison Break Problem

- All of you are prisoners
- Infinitely often a guard picks uniformly at random and sends only one of you to a room for few minutes.
- This room only has an on-off light switch initially turned off (that the guard does not touch).
- If someone says to the guard that everybody entered the room and he is right, then the guard will set you free.
- If he is wrong, you will be locked up forever.
- Now is the only time you can cooperate to find a strategy.
- What is this strategy? (Do not give the answer yet)

II. Populations

- The population is threatened if more than a threshold dies on the way back to the sea
- If a penguin starts its trip with a low temperature, the probability that it reaches the sea is low
The Penguins Algorithm

- Each penguin is given a device with
  - A temperature sensor
  - A finite counter (<6) initially 1 (resp. 0) if temperature is low (resp. high)
  - Bluetooth: devices exchange values if close
  - 3G capability: devices send an alert if ≥5 values are 1

- Every now and then each pair of penguins meet

The Penguins Algorithm

- When two devices meet, one keeps in its counter the sum of the values whereas the other puts it back to 0
- The first device that has value 5 triggers the alert

0 0 1 0 1 1 0 1 0 1
0 0 2 0 0 1 0 1 0 1
0 0 0 0 0 3 0 1 0 1
The Penguins Algorithm

- When two devices meet, one keeps in its counter the sum of the values whereas the other puts it back to 0
- The first device that has value 5 triggers the alert

Algorithm "+"

- Determining if the numbers of C's is the sum of the number of A's and the number of B's
- When a A meets a B, it cancels the B and raises a flag; when it later meets a C, it cancels itself and the C

Population Protocols

- Every agent has a bounded memory independent of the size of the system
- Algorithms are uniform and anonymous
- 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)
Algorithm “or”

- Every agent has the following input/output/state:
  - Input $\in \{0,1\}$
  - Output $\in \{0,1\}$
  - State $\in \{0,1\}$
- Transition: every agent with state 0 that meets an agent with state 1 changes its value to 1:
  - $\{0,1\} \rightarrow \{1,1\}$

Algorithm “>”

- How to know whether there are more girls than boys at a party?
- Input $\in \{g,b\}$; Output $\in \{yes,no\}$
- State $\in \{g,b,\text{yes},\text{no}\}$
- Transitions:
  - $(g,b) \rightarrow (\text{no},\text{no})$
  - $(g,\text{no}) \rightarrow (g,\text{yes})$
  - $(b,\text{yes}) \rightarrow (b,\text{no})$
  - $(\text{yes,\no}) \rightarrow (\text{no,\no})$

The Majority Algorithm

$\{g,b\} \rightarrow \{\text{no,\no}\}$, $\{g,\text{no}\} \rightarrow \{g,\text{\yes}\}$,
$(b,\text{yes}) \rightarrow (b,\text{\no})$, $(\text{yes,\no}) \rightarrow (\text{\no,\no})$

The Majority Algorithm

$\{g,b\} \rightarrow \{\text{no,\no}\}$, $\{g,\text{no}\} \rightarrow \{g,\text{\yes}\}$,
$(b,\text{yes}) \rightarrow (b,\text{\no})$, $(\text{yes,\no}) \rightarrow (\text{\no,\no})$
The Majority Algorithm

\[(g,b) \rightarrow (no,no), (g,no) \rightarrow (g,yes),
(b,yes) \rightarrow (b,no), (yes,no) \rightarrow (no,no)\]

The Majority Algorithm

\[(g,b) \rightarrow (no,no), (g,no) \rightarrow (g,yes),
(b,yes) \rightarrow (b,no), (yes,no) \rightarrow (no,no)\]
The Majority Algorithm

(g,b) -> (no,no), (g,no) -> (g,yes),
(b,yes) -> (b,no), (yes,no) -> (no,no)

The Majority Algorithm

(g,b) -> (no,no), (g,no) -> (g,yes),
(b,yes) -> (b,no), (yes,no) -> (no,no)

The Majority Algorithm

(g,b) -> (no,no), (g,no) -> (g,yes),
(b,yes) -> (b,no), (yes,no) -> (no,no)
The Majority Algorithm

\((g,b) \rightarrow (\text{no},\text{no}),\ (g,\text{no}) \rightarrow (g,\text{yes}),\ \ (b,\text{yes}) \rightarrow (b,\text{no}),\ (\text{yes},\text{no}) \rightarrow (\text{no},\text{no})\)

The Majority Algorithm

\((g,b) \rightarrow (\text{no},\text{no}),\ (g,\text{no}) \rightarrow (g,\text{yes}),\ \ (b,\text{yes}) \rightarrow (b,\text{no}),\ (\text{yes},\text{no}) \rightarrow (\text{no},\text{no})\)

The Majority Algorithm

\((g,b) \rightarrow (\text{no},\text{no}),\ (g,\text{no}) \rightarrow (g,\text{yes}),\ \ (b,\text{yes}) \rightarrow (b,\text{no}),\ (\text{yes},\text{no}) \rightarrow (\text{no},\text{no})\)

The Majority Algorithm

\((g,b) \rightarrow (\text{no},\text{no}),\ (g,\text{no}) \rightarrow (g,\text{yes}),\ \ (b,\text{yes}) \rightarrow (b,\text{no}),\ (\text{yes},\text{no}) \rightarrow (\text{no},\text{no})\)

Multiplication

- Determining if the number of Cs is the product of the number of As and the number of Bs

- Whenever a A sees a new B, it cancels a C: problem - what is a new B?

Result

**Theorem 1:** population protocols compute exactly first order Presburger's arithmetic: +, -, ≤, >, or, not, and, ...

The Crash Problem

- Let's take a look at the Penguins Algorithm if one dies
The Crash Problem

• Let's take a look at the Penguins Algorithm if one dies
• The alarm will not be triggered if 5 is not reached

When two agents communicate, one acts as the initiator for O1 and the other as the initiator for O2.

Crash Tolerant Algorithm

• Every agent performs twice the original algorithm: O1 and O2
• When two agents communicate, one acts as the initiator for O1 and the other as the initiator for O2

The alarm will not be triggered if 5 is not reached.
When two agents communicate, one acts as the initiator for O1 and the other as the initiator for O2.

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

![Diagram of the Crash Tolerant Algorithm](image1.png)

Even if one dies, the alarm is still triggered by the other one whose values reached 5:0 or 0:5.

![Diagram of the Crash Tolerant Algorithm](image2.png)
Result

*Theorem 2: population protocols compute exactly Presburger’s arithmetic with a constant number of crashes*

NB. The notion of “computation” is slightly revised

The Privacy Problem

- 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 strictly more do?

- 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

- How can we hide the initial values from curious agents?

Obfuscation

*Idea:* each agent sends different values to different agents

Each agent maintains a <value, bit> pair

Agents can change their values when they meet:

\[(v1,1), (v2,1) \rightarrow (v1+1,1), (v2-1,1)\]

They use other rules to converge to a solution:

\[(*,1), (*,*) \rightarrow (*,0), (*,*) ; (*,0), (*,1) \rightarrow (*,1), (*,1)\]

\[(v1,0), (v2,0) \rightarrow (v1+v2,0), (0,0) ; (v1,0), (0,0) \rightarrow (v1,0), (1,0)\]

Privacy Preserving Algorithm

\[(*,1), (*,*) \rightarrow (*,0), (*,*) ; (*,0), (*,1) \rightarrow (*,1), (*,1)\]

\[(v1,0), (v2,0) \rightarrow (v1+v2,0), (0,0) ; (v1,0), (0,0) \rightarrow (v1,0), (1,0)\]

Privacy Preserving Algorithm

\[(*,1), (*,*) \rightarrow (*,0), (*,*) ; (*,0), (*,1) \rightarrow (*,1), (*,1)\]

\[(v1,0), (v2,0) \rightarrow (v1+v2,0), (0,0) ; (v1,0), (0,0) \rightarrow (v1,0), (1,0)\]
Privacy Preserving Algorithm

\( (*,1),(*,*) \rightarrow (*,0),(*,*) \); \( (*,0),(*,1) \rightarrow (*,1),(*,1) \)

\( (v_1,0),(v_2,0) \rightarrow (v_1+v_2,0),(0,0) \); \( (v_1,0),(0,0) \rightarrow (v_1,0),(\bot,0) \)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

Privacy Preserving Algorithm

\( (*,1),(*,*) \rightarrow (*,0),(*,*) \); \( (*,0),(*,1) \rightarrow (*,1),(*,1) \)

\( (v_1,0),(v_2,0) \rightarrow (v_1+v_2,0),(0,0) \); \( (v_1,0),(0,0) \rightarrow (v_1,0),(\bot,0) \)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

Privacy Preserving Algorithm

\( (*,1),(*,*) \rightarrow (*,0),(*,*) \); \( (*,0),(*,1) \rightarrow (*,1),(*,1) \)

\( (v_1,0),(v_2,0) \rightarrow (v_1+v_2,0),(0,0) \); \( (v_1,0),(0,0) \rightarrow (v_1,0),(\bot,0) \)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

Privacy Preserving Algorithm

\( (*,1),(*,*) \rightarrow (*,0),(*,*) \); \( (*,0),(*,1) \rightarrow (*,1),(*,1) \)

\( (v_1,0),(v_2,0) \rightarrow (v_1+v_2,0),(0,0) \); \( (v_1,0),(0,0) \rightarrow (v_1,0),(\bot,0) \)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

Privacy Preserving Algorithm

\( (*,1),(*,*) \rightarrow (*,0),(*,*) \); \( (*,0),(*,1) \rightarrow (*,1),(*,1) \)

\( (v_1,0),(v_2,0) \rightarrow (v_1+v_2,0),(0,0) \); \( (v_1,0),(0,0) \rightarrow (v_1,0),(\bot,0) \)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>
Privacy Preserving Algorithm

\((*,1),(*,*) \rightarrow (*,0),(*,*) \); \((*,0),(*,1) \rightarrow (*,1),(*,1)\)
\((v_1,0), (v_2,0) \rightarrow (v_1+v_2,0), (0,0) \); \((v_1,0),(0,0) \rightarrow (v_1,0), (\bot,0)\)

\begin{array}{cccc}
4,1 & 3,1 & 4,1 & 3,1 \\
4,0 & 3,1 & 4,1 & 3,1 \\
4,0 & 3,0 & 4,1 & 3,1 \\
7,0 & 0,0 & 4,1 & 3,1 \\
8,0 & 0,0 & 4,1 & 3,1 \\
\end{array}

Privacy Preserving Algorithm

\((*,1),(*,*) \rightarrow (*,0),(*,*) \); \((*,0),(*,1) \rightarrow (*,1),(*,1)\)
\((v_1,0), (v_2,0) \rightarrow (v_1+v_2,0), (0,0) \); \((v_1,0),(0,0) \rightarrow (v_1,0), (\bot,0)\)

\begin{array}{cccc}
8,0 & 1,0 & 4,1 & 3,1 \\
8,0 & 1,0 & 4,1 & 3,1 \\
8,0 & 1,0 & 4,1 & 3,1 \\
\end{array}

Privacy Preserving Algorithm

\((*,1),(*,*) \rightarrow (*,0),(*,*) \); \((*,0),(*,1) \rightarrow (*,1),(*,1)\)
\((v_1,0), (v_2,0) \rightarrow (v_1+v_2,0), (0,0) \); \((v_1,0),(0,0) \rightarrow (v_1,0), (\bot,0)\)

\begin{array}{cccc}
8,0 & 1,0 & 4,1 & 3,1 \\
8,0 & 1,0 & 4,1 & 3,1 \\
8,0 & 1,0 & 4,1 & 3,1 \\
\end{array}

Privacy Preserving Algorithm

\((*,1),(*,*) \rightarrow (*,0),(*,*) \); \((*,0),(*,1) \rightarrow (*,1),(*,1)\)
\((v_1,0), (v_2,0) \rightarrow (v_1+v_2,0), (0,0) \); \((v_1,0),(0,0) \rightarrow (v_1,0), (\bot,0)\)

\begin{array}{cccc}
8,0 & 1,0 & 4,1 & 3,1 \\
8,0 & 1,0 & 4,1 & 3,1 \\
8,0 & 1,0 & 4,1 & 3,1 \\
\end{array}
**Privacy Preserving Algorithm**

\[
\begin{align*}
(*,1),(\cdot, *) &\rightarrow (*,0),(\cdot, *) ; (*,0),(\cdot, *) \rightarrow (*,1),(\cdot, *) \\
(v1,0),(v2,0) &\rightarrow (v1+v2,0),(0,0) ; (v1,0),(0,0)-->(v1,0),(_\bot,0)
\end{align*}
\]

\[
\begin{array}{cccc}
8.0 & 4.1 & 3.1 & 0.0 \\
8.0 & 4.1 & 3.1 & _\bot \phantom{0} \\
8.0 & 4.0 & 3.0 & _\bot \\
15.0 & _\bot & 0.0 & _\bot \\
\end{array}
\]

**Theorem 3:** Population protocols can privately compute exactly first order Presburger’s arithmetic.
III. Concurrency

Before: Computers used a single processor

Uniprocessor Machine

Nowadays: Computers use multiple processors

Multicore Machine

Buying a new machine used to speedup programs

Program

Old uniprocessor

Speedup

1.8x
3.6x
7x

Multi-core Machine

Concurrency is the only way to speedup programs

Assume we have an old uniprocessor machine

Multiprocessor

5/06/15 Vincent Gramoli - CS4HS 2014

How fast can we sort 16 numbers?

Assume we have an old uniprocessor machine

Multi-core Machine

Buying a new machine used to speedup programs

Program

Old uniprocessor

Speedup

1.8x
3.6x
7x

Multi-core Machine

Concurrency is the only way to speedup programs

Assume we have an old uniprocessor machine

Multiprocessor

5/06/15 Vincent Gramoli - CS4HS 2014
How fast can we sort 16 numbers?

1. The processor chooses a specific number, the pivot

2. The processor puts lower numbers on the left and higher numbers on the right of the pivot
How fast can we sort 16 numbers?
How fast can we sort 16 numbers?

4 6 8 2 10 12 13 11 7 3 14 9 15 1 16

The processor compares 15 numbers to the pivot

3. It takes at least 15 comparisons to get there, but we must do the same for the new left and right parts now... and so on

Assume we have now a 8-core machine

Step 1
Step 2
Step 3
Step 4
Step 5
Step 6
Step 7
Step 8
Step 9

Cores do the vertical comparisons of one box in parallel

How fast can we sort 16 numbers?

Theorem 5. Sorting $n$ nodes with $n/2$ processors takes $O(\log^2 n)$ steps.

Amount of numbers to sort

Time to sort them with one processor

Time to sort them with $n/2$ processors

Solution to the Prisoners Problem

Conclusion

• Distributed computing is about concurrency

• Distributed agents/processors act independently

• Together they can be more efficient than any one of them

• They need to synchronize sometimes by proceeding in steps or by conveying information to a leader

• Distributed computing algorithms are not trivial: if you couldn’t find a solution to the prisoners problem during this talk, it is probably due to the difficulty of doing multiple things concurrently

References


