Unison : clocks synchronization

Each process $u$ has a clock : $c(u)$
- the difference of clock values between neighbors is at most 1 (safety)
- Each process increments its clock infinitely often (liveness)

Distributed algorithm using a bounded clock – a bound on network size is known ($K$)

∀ $v$ in the neighborhood of $u$, we have $c(v) \in \{c(u), c(u)+1\%K\}$
→ $c(u) := (c(u)+1) \%K$

The initial configuration is a safe configuration

$K > \text{size of networks}$
Unison: clocks synchronization

∀v in the neighborhood of u, we have c(v) ∈ [c(u), c(u)+1%K]
→ c(u) := (c(u)+1) %K

The initial configuration is a safe configuration

K > size of networks

Unison: clocks synchronization

∀v in the neighborhood of u, we have c(v) ∈ [c(u), c(u)+1%K]
→ c(u) := (c(u)+1) %K

The initial configuration is a safe configuration

K > size of networks

Unison: clocks synchronization

∀v in the neighborhood of u, we have c(v) ∈ [c(u), c(u)+1%K]
→ c(u) := (c(u)+1) %K

The initial configuration is a safe configuration

K > size of networks

Unison: clocks synchronization

∀v in the neighborhood of u, we have c(v) ∈ [c(u), c(u)+1%K]
→ c(u) := (c(u)+1) %K

The initial configuration is a safe configuration

K > size of networks

Unison: clocks synchronization

∀v in the neighborhood of u, we have c(v) ∈ [c(u), c(u)+1%K]
→ c(u) := (c(u)+1) %K

The initial configuration is a safe configuration

K > size of networks

Unison: clocks synchronization

∀v in the neighborhood of u, we have c(v) ∈ [c(u), c(u)+1%K]
→ c(u) := (c(u)+1) %K

The initial configuration is a safe configuration

K > size of networks
Unison: clocks synchronization
\[ \forall v \text{ in the neighborhood of } u, \text{ we have } c(v) \in [c(u), (c(u)+1) \% K] \]
\[ \implies c(u) := (c(u)+1) \% K \]

The initial configuration is a safe configuration

K > size of networks

Self-stabilizing algorithm

Goal of the algorithm

To rewind (To reset) To reuse

Correct Configuration

Distributed Algorithm

Unison algorithm is not self-stabilizing
To rewind (To reset) To reuse
Chaos to reach a correct configuration
Goal of the algorithm
Distributed Algorithm

- Awerbuch, Patt-Shamir, Varghese: self-stabilization by local checking and global reset (extended abstract). FOCS '91
- Awerbuch, Patt-Shamir, Varghese, Dolev: self-stabilization by local checking and global reset (extended abstract). WDAG'94

Self-stabilizing resetting algorithm

- identified network
- mono initiator
- weakly fair scheduler

SDR algorithm - Self-stabilizing Distributed Reset algorithm
- anonymous network,
- no network knowledge
- multi initiators
- unfair scheduler
- linear number of process moves
- Unbounded timestamps

Self stabilizing resetting

Total mess
Chaos
Correct Configuration
reseting

Chaos → Correct behavior

I : distributed algorithm solving a task (static or dynamic) from a correct configuration
Chaos
SDR ◦ I

Correct configuration of I

I: distributed algorithm solving a task (static or dynamic) from a correct configuration

Correct configuration of I
P_ICorrect(u): predicate on u and u's neighbor variables
P_ICorrect(u) is closed along I
∀ u we have (P_ICorrect(u) == true) = the configuration is correct (≠ legitimate)
Pr(u) is closed along I iff Pr(u) stays verified along any execution of I

Local resetting in I

I: distributed algorithm solving a task (static or dynamic) from a correct configuration

Local resetting in I
reset(u): macro resetting the value of u's variables
P_reset(u): predicate on u's variable:
   it is true iff u's variable are reseted
If u and all u's neighbors verify P_reset then
P_ICorrect(u) == true
Unison – resetable?

\( \forall v \text{ in the neighborhood of } u, \text{ we have } c(v) \in \{c(u), c(u)+1\%K\} \)
\( \rightarrow c(u) := (c(u)+1) \%K \)

\( P_{\text{ICorrect}}(u) \) is the safety predicate

\( \forall v \text{ in the neighborhood of } u, \text{ we have } c(v) \in \{c(u)-1\%K, c(u), c(u)+1\%K\} \)

\( \text{reset}(u) : c(u) := 0 \)

\( P_{\text{reset}}(u) : c(u) = 0 \)

If \( u \) and all \( u \)'s neighbors verify \( c==0 \) then
\( P_{\text{ICorrect}}(u)==true \)

\( K > \text{size of networks} \)

SDR – Overall presentation

• A process \( u \) starts the resetting because
\( P_{\text{ICorrect}}(u) != true \)

• The resetting is propagated (a DAG rooted at \( u \) is built)

• The DAG is frozen from the leaves to the initiator of the resetting (\( u \))

• Processes go back to the Initial algorithm/task from the initiator (\( u \)) to the leaves of the DAG

SDR – variables on \( u \)

• To store the DAG structure:
  – \( d(u) \) : distance to an initiator of the resetting

SDR – variables on \( u \)

• To store the DAG structure:
  – \( d(u) \) : distance to an initiator of the resetting

SDR – variables on \( u \)

• To store the DAG structure:
  – \( d(u) \) : distance to an initiator of the resetting

• The status of the resetting : \( st(u) \) : RB, RF, or C
  – a resetting is in progress
    • RB : propagation of the resetting
    • RF : propagation of the ending of broadcast phase
  – no resetting is in progress : C
**SDR – Self-stabilizing Distributed Reset**

**R_R** : a process $u$ starts the reseting because

$$P_{\text{-ICorrect}}(u) \neq \text{true} \text{ or } ...$$

**R_R** : If $(st(u)==C)$ and $\neg P_{\text{-ICorrect}}(u)$

$$\text{reset}(u); \ st(u) := \text{RB}; \ d(u) := 0;$$

A process performs at most one time $R_R$ rule

A process can be the root of a single DAG

At most $n$ DAG structure is built during any execution

($n$ being the network size)

---

**R_B** : If $(st(u)==C)$ and a’ $u$ neighbor $v$ verified $(st(v)==\text{RB})$

$$\text{reset}(u); \ st(u) := \text{RB}; \ d(u) := d(v)+1;$$

**R_B** : the reseting is propagated (a DAG is built)

A process cannot join two times the same DAG structure

A process performs at most $n-1$ times the rule $R_B$

($n$ being the network size)

---

**R_F** : If $(st(u)==\text{RB})$ and the reseting propagation is over in the DAG rooted at $u$

$$\text{reset}(u); \ st(u) := \text{RF};$$

**R_F** : the DAG is frozen from the leaves to the initiator

A process takes the status $\text{RB}$ at most $n$ times

A process performs at most $n+1$ times the rule $R_F$

($n$ being the network size)

---

**Unison : clocks synchronization**

$$\forall v \text{ in the neighborhood of } u, \text{ we have } c(v) \in \{c(u), c(u)+1\text{mod}K\}$$

$$\rightarrow c(u) := (c(u)+1) \text{ mod} K$$

An Unsafe Configuration

---

**Unison : clocks synchronization**

$$\forall v \text{ in the neighborhood of } u, \text{ we have } c(v) \in \{c(u), c(u)+1\text{mod}K\}$$

$$\rightarrow c(u) := (c(u)+1) \text{ mod} K$$

Resetting Propagation phase

---

**A process**

- A process performs at most one time $R_R$ rule
- A process can be the root of a single DAG
- At most $n$ DAG structure is built during any execution ($n$ being the network size)

**A process cannot**

- Join two times the same DAG structure
- Performs at most $n-1$ times the rule $R_B$

($n$ being the network size)

**A process takes**

- The status $\text{RB}$ at most $n$ times
- Performs at most $n+1$ times the rule $R_F$

($n$ being the network size)

---

**Reseting Propagation phase**

- $d$ takes the status $\text{RB}$ at most $n$ times
- Performs at most $n$ times the rule $R_C$

($n$ being the network size)
∀v in the neighborhood of u, we have c(v) ∈ {c(u), c(u)+1%K} → c(u) := (c(u)+1) %K

Resetting Propagation phase

End of Resetting Propagation phase

Freezing phase
∀v in the neighborhood of u, we have c(v) ∈ \{c(u), c(u)+1\%K\} → c(u) := (c(u)+1) \%K

Freezing phase

End of Freezing phase

Back to the Initial Algorithm
∀v in the neighborhood of u, we have c(v) ∈ {c(u), c(u)+1%K} → c(u) := (c(u)+1) %K

A Safe Configuration

Step/Round - illustration

1st round - 3 steps

2nd round – 2 steps

enabled
triggered
unable
### Self-stabilizing Unison

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Memory</th>
<th>Stops</th>
<th>Rounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1]</td>
<td>$O(n^3)$</td>
<td>-</td>
<td>$O(Dn)$ [Boulinier PhD]</td>
</tr>
<tr>
<td>[2]</td>
<td>$O(n)$</td>
<td>$O(Dn^3)$ [3]</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>SDR o I</td>
<td>unbounded</td>
<td>$O(Dn^2)$</td>
<td>$3n$</td>
</tr>
</tbody>
</table>

[1] Couvreur, Francez, Gouda: Asynchronous unison (extended abstract), ICDCS’92

---

**Technical report on HAL:**
Self Stabilizing Distributed Cooperative Reset.

*ICDCS’19*