Self-stabilizing Systems

Ulf Nilsson
TCSLAB, IDA, LiU

Overview

General idea of self-stabilization

- A distributed system with the ability to automatically recover to normal behavior in case of (transient) faults.
- We model systems optimistically, i.e. we describe the correct behavior of the system. In case of deviations from correct behavior the system is guaranteed to return to its correct behavior in finite time.
- Data and program counter may be corrupted at any time, but program is non-corrupted.
- No centralized control.

Pros and cons

Advantages:

+ Fault tolerance
+ Initialization
+ Adaptation to topological changes

Disadvantages:

– Initial inconsistencies
– High complexity
– Local unawareness of stabilization

Basic Notions

Computation model

Definition A system is a pair $(C, \rightarrow)$ where C is a set of configurations and $\rightarrow \subseteq C \times C$ a transition relation. An execution is a maximal sequence $c_0, c_1, c_2, \ldots$ such that $c_{i-1} \rightarrow c_i$ for each $i > 0$.

Note: Initial configuration is irrelevant.

Note: Interleaving model.

Definition of self-stabilization

Definition A set $\mathcal{L} \subseteq C$ is called legitimate (relative some property P) if every execution starting from some $c \in \mathcal{L}$ has property P. (Alternatively, every $c \in \mathcal{L}$ has property P.)

Definition A system $(C, \rightarrow)$ is self-stabilizing if every execution $c_0, c_1, c_2, \ldots$

- eventually reaches $\mathcal{L}$, i.e. there is some $i \geq 0$ such that $c_i \in \mathcal{L}$ (liveness property);
- the set $\mathcal{L}$ is closed under $\rightarrow$, i.e. if $c_i \in \mathcal{L}$ and $c_i \rightarrow c_{i+1}$ then $c_{i+1} \in \mathcal{L}$ (safety property).

Definition of self-stabilization (cont)

Sometimes also:

- **No deadlock**: There is always some next configuration (all executions are infinite);
- **Fairness**: Each legitimate configuration is reached an infinite number of times in each execution;

Proving Self-stabilization

General idea: Exhibit a norm function $f : C \to W$ where W is a well-founded set (typically the natural numbers) such that for each transition $c_i \to c_{i+1}$ either $c_i \in \mathcal{L}$ or $f(c_i) > f(c_{i+1})$.

Theorem Let $(C, \to)$ be a system. If all terminal configurations are in $\mathcal{L}$, $\mathcal{L}$ is closed under $\to$ and there is a norm function $f : C \to W$ then the system is self-stabilizing.

Note: C.f. proof of termination.

Examples

Dijkstra's algorithm I

Consider a ring of n processes $P_0, \ldots, P_{n-1}$.

- each process has a finite number (=K) states;
- each process communicates with its left neighbor (P_0 with P_{n-1});
- a process can only change its own state;

Objective:

- At most one process may use some critical resource (mutual exclusion).

Therefore:

- Make sure that exactly one process can change its state.

Dijkstra's algorithm II

Assume n tasks $P_0, \ldots, P_{n-1}$ in an undirected ring.

P_0:
1. do forever
2. if $x_0 = x_{n-1}$ then
3. $x_0 := (x_0 + 1)$ mod K (where $K > n$)

P_i: $(0 < i < n)$
1. do forever
2. if $x_i \neq x_{i-1}$ then
3. $x_i := x_{i-1}$

Parametric systems

- Proving self-stabilization for a specific system is easy (finite state problem.)
- Want to prove stabilization for all instances.
- Modulo some topology.

Dijkstra's algorithm III

$c \in \mathcal{L}$ iff exactly one process in c may change its state.

Lemma $\mathcal{L}$ is closed under Dijkstra's algorithm.

Lemma Dijkstra's algorithm converges to $\mathcal{L}$.

Corollary Dijkstra's algorithm converges to mutual exclusion.

Note: Non-stabilizing if $K < n - 1$

Ghosh's four state algorithm I

For P_0:
1. do forever
2. if $x_0 x_1 = 12$ then $x_0 := 3$
3. if $x_0 x_1 = 30$ then $x_0 := 1$

For P_{n-1}:
1. do forever
2. if $x_{n-2} x_{n-1} = 32$ then $x_{n-1} := 0$
3. if $x_{n-2} x_{n-1} = 10$ then $x_{n-1} := 2$

For P_i: $(0 < i < n - 1)$
1. do forever
2. if $x_i + 1 = x_{i-1}$ then $x_i := x_{i-1}$
3. if $x_i + 1 = x_{i+1}$ then $x_i := x_{i+1}$

Note: Addition modulo 4.

Ghosh's four state algorithm II

Legitimate states $\mathcal{L} = \{1^+, 3^+\}\{0^+, 2^+\}$

Example run:

$$\begin{aligned} 13230 &\to 13330 \to 13300 \\ &\to 13000 \to 10000 \\ &\to 11000 \to 11100 \\ &\to 11110 \to 11112 \\ &\to 11122 \to 11222 \end{aligned}$$

Spanning tree I

Link-register model:

- Communication by registers R_{ij}
- P_i writes in R_{ij}
- P_j only reads from R_{ij}
- $R_{ij}.parent = 1$ if j is the parent of i
- $R_{ij}.dis$: distance from P_i to root
- $N(i)$ is the set of i's neighbors

Each process P_j has a local copy lR_{ij} of R_{ij}.

Spanning tree II

The root process, P_i:

1. do forever
2. for each $m \in N(i)$ do $R_{im} := (0, 0)$

Other processes, P_i:

1. do forever
2. for each $m \in N(i)$ do $lR_{mi} := R_{mi}$
3. FirstFound := false
4. dist := $1 + min\{lR_{mi}.dis \mid m \in N(i)\}$
5. for each $m \in N(i)$ do
6. if not FirstFound and $lR_{mi}.dis$ =dist-1 then
7. R_{im} := (1,dist)
8. FirstFound := true
9. else R_{im} := (0,dist)

Uniform systems

- Ideally all machines execute the same algorithm.
- Generally not possible.
- Algorithm(s) independent of n (number of machines).
- Dijkstra's algorithm requires that $K \geq n - 1$.

Maximal matching I

- A matching in an (undirected) graph is a subset of edges such that no node is incident to more than one edge.
- A matching is maximal if it cannot be extended with more edges (NB: no need for maximal cardinality).
- Each node is a process P_i
- Each P_i has a variable $ptr_i \in N(i) \cup \{null\}$

Maximal matching II

1. do forever
2. if $ptr_i = null \wedge (\exists P_j \in N(i) \mid ptr_j = i)$ then
3. $ptr_i = j$
4. if $ptr_i = null \wedge (\forall P_j \in N(i) \mid ptr_j \neq i) \wedge$
5. $(\exists P_j \in N(i) \mid ptr_j = null)$ then
6. $ptr_i = j$
7. if $ptr_i = j \wedge ptr_j = k \wedge k \neq i$ then
8. $ptr_i = null$

Methodology

Fair composition of systems

Composition of $\mathcal{AL}_1$ and $\mathcal{AL}_2$:

- Let $\mathcal{AL}_1$ be an algorithm over a state space S_1 (the server process)
- Let $\mathcal{AL}_2$ be an algorithm over a state space $S_1 \times S_2$ (the client process).
- $\mathcal{AL}_2$ may read S_1 but write only in S_2.
- Interleave the two algorithms (fairly).

If $\mathcal{AL}_1$ and $\mathcal{AL}_2$ are self-stabilizing then so is the composition of $\mathcal{AL}_1$ and $\mathcal{AL}_2$.

Mutual exclusion (revisited)

- Mutual exclusion in a arbitrary (connected) topology
- Special root processor
- Construct a spanning tree
- Depth-first traversal of the tree is a virtual ring
- Apply Dijkstra's algorithm to virtual ring

Mutual exclusion in tree I

Root process:

1. do forever
2. $lr_{1,i} := r_{1,i}$
3. if $lr_{\delta,i} = r_{i,1}$ then
4. $r_{i,2} := (lr_{1,i} + 1)$ mod $(4n - 5)$
5. for $m := 2$ to δ do
6. $lr_{m,i} := r_{m,i}$
7. $r_{i,m+1} := lr_{m,i}$
8. od
9. od

Mutual exclusion in tree II

Non-root processes:

1. do forever
2. $lr_{1,i} := r_{1,i}$
3. if $lr_{1,i} \neq r_{i,2}$ then
4. $r_{i,2} := lr_{1,i}$
5. for $m := 2$ to δ do
6. $lr_{m,i} := r_{m,i}$
7. $r_{i,m+1} := lr_{m,i}$
8. od
9. od

Further issues

- Pseudo-stabilization
- Converting non-stabilizing algorithms into stabilizing onces
- Randomized algorithms
- Conversion between models