A Model for Large Scale Self-Stabilization

Thomas Herault, Pierre Lemarinier, Olivier Peres,
Laurence Pilard, Joffroy Beauquier

LRI, Université Paris-Sud

Cédric Tedeschi - WG GRAAL - May 31, 2007

A technique and some platforms

- *Peer-to-peer* networks and grids
  - large scale
  - every pair of nodes are able to communicate
  - dynamic set of *neighbors*
  - unstable platforms (crashes)
- *Self-stabilizing* algorithms
  - small scale
  - designed for distributed systems with a static topology
  - fixed set of links
- Need for overcoming this division
  - new model abstracting P2P platforms injected in self-stabilization
  - resource discovery service ⟶ dynamic neighborhood
  - failure detection service ⟶ crash awareness

A technique and some platforms

- *Peer-to-peer* networks and grids
  - large scale
  - every pair of nodes are able to communicate
  - dynamic set of *neighbors*
  - unstable platforms (crashes)
- *Self-stabilizing* algorithms
  - small scale
  - designed for distributed systems with a static topology
  - fixed set of links
- Need for overcoming this division
  - new model abstracting P2P platforms injected in self-stabilization
  - resource discovery service $\longrightarrow$ dynamic neighborhood
  - failure detection service $\longrightarrow$ crash awareness

A technique and some platforms

- *Peer-to-peer* networks and grids
  - large scale
  - every pair of nodes are able to communicate
  - dynamic set of *neighbors*
  - unstable platforms (crashes)
- *Self-stabilizing* algorithms
  - small scale
  - designed for distributed systems with a static topology
  - fixed set of links
- Need for overcoming this division
  - new model abstracting P2P platforms injected in self-stabilization
  - resource discovery service $\longrightarrow$ dynamic neighborhood
  - failure detection service $\longrightarrow$ crash awareness

Outline

Outline

Model - definitions (1)

A *P2P oriented* model

- No fixed topology, no set of communication links (too large)
- Physical layer abstracted, *neighborhood* based on resource discovery
- $(\mathcal{I}, <)$, the totally ordered set of process identifiers
- $P \subseteq \mathcal{I}$, the set of *correct* processes
- $\mathcal{C} = \{c_{a \to b} | \forall a, b \in \mathcal{I}^2\}$, the set of possible FIFO channels

State - configuration

- The *state* of a process is the set of its variables and their values
- The *state* of a channel is the ordered list of the messages it contains
- The *configuration* of a system is the product of the states of every $i \in \mathcal{I}$ and every $c \in \mathcal{C}$

Model - definitions (2)

Execution

An *execution* is a sequence $C_1, A_1, C_2, A_2, \ldots, C_i, A_i, \ldots$ such that $\forall i \in \mathbb{N}^*$, applying transition A_i to configuration C_i yields to configuration C_{i+1}

Self-stabilization

Let $\mathcal{L}$ a set of configurations (satisfying some properties, and defining what is a *stable* configuration). An algorithm is self-stabilizing to $\mathcal{L}$ if and only if :

- **Correctness** Every execution starting from a configuration of $\mathcal{L}$ verifies the specification
- **Closure** Every configuration of all executions starting from a configuration of $\mathcal{L}$ is a configuration of $\mathcal{L}$
- **Convergence** Starting from any configuration, every execution reaches a configuration of $\mathcal{L}$.

Model - services

Resource discovery

- Oracle providing identifers in $\mathcal{I}$:
  - assumes an *id* eventually returned
  - example : enumerates $\mathcal{I}$ in an infinite loop

Failure detection

- Match the self-stabilization paradigm
  - valid behavior, most of the time
  - infrequent transient failures
- Model : arbitrary initialization and then failure-free run (*i.e., all detectors converge eventually*)
- Implementation : distributed failure detector
  - function *suspect* : $\mathcal{I} \rightarrow$ *boolean*
  - after a finite time return *true iff* the *id* $\notin P$ from then on.

Model - execution

Algorithm

- Each node executes the same code
- set of guarded rules $<$ *guard* $>\rightarrow<$ *statement* $>$
- $<$ *guard* $>$: boolean expression (variables and incoming message)
- $<$ *statement* $>$:
  - consumes the message (if any)
  - modifies the local state
  - sends messages

Scheduler

Each statement is eventually triggered if the guard is infinitely true

Outline

The algorithm - principles

- Topology kept free of cycle by *heap invariant*
  - id_p must be lower than the *id* of the father of p
  - id_p must be greater than the *id* of any of its children
- Every process checks consistency in its neighborhood
  - using the failure detector to eliminate stopped processes
  - its parent considers it as a child
  - its children consider it as their parent
- Each process being a root ($parent_p = id_p$)
  - connect new processes via the resource discovery
  - enforce the global invariant

The algorithm - details (1)

Algorithm - Constants, variables, messages

- Constants :
  - id_p
  - δ
- variables :
  - $parent_p$
  - $children_p$
- Messages :
  - $Exists(id)$
  - $YouAreMyChild(id)$
  - $Neighbor?(id)$
  - $NotNeighbor(id)$

The algorithm - details (1)

Algorithm - Procedures and functions

- $Neighborhood(p)$: $return\{id_q \in children_p \cup \{parent_p\}\} \setminus \{id_p\}$
- $Sanity\_check(p)$:
  IF $parent_p < id_p$ THEN $parent_p := id_p$
  IF $|children_p| > \delta$ THEN $children_p := \emptyset$
  $children_p := \{id_q \in children_p / id_q < id_p\}$
- $Suspect(id_p)$
- $Detect\_failures(p)$:
  IF $parent_p \neq id_p \wedge Suspect(parent_p)$ THEN $parent_p = id_p$
  $\forall id_q \in children_p$ IF $Suspect(id_q)$ THEN $children_p = children_p \setminus \{id_q\}$
- $RD\_Get()$

The algorithm - details (2)

True $\rightarrow$

$Sanity\_check(p)$; $Detect\_failures(p)$

$\forall id_q \in Neighborhood(p)$ SEND $Neighbor?(id_p)$ TO q

IF $parent_p = id_p$ THEN

$id_q := RD\_Get()$

IF $id_q > id_p$ THEN SEND $Exists(id_p)$ TO q

The algorithm - details (2)

Reception of $Neighbor?(id_q) \rightarrow$

$Sanity\_check(p)$

IF $id_p < id_q$ THEN

IF $parent_p = id_p$ THEN $parent_p := id_q$

ELSE IF $id_q \notin children_p$ THEN

IF $|children_p| < \delta$ OR $(|children_p| = \delta$ AND $\exists id_r | id_r < id_q)$ THEN

$children_p := children_p \setminus id_r \cup \{id_q\}$

ELSE IF $id_p \neq id_q$ THEN

SEND $NotNeighbor(id_p)$ TO q

The algorithm - details (2)

Reception of $NotNeighbor(id_q) \rightarrow$

$Sanity\_check(p)$

IF $parent_p = id_q$ THEN $parent_p : id_p$

$children_p = children_p \setminus \{id_q\}$

The algorithm - details (2)

Reception of $Exists(id_q) \rightarrow$

$Sanity\_check(p)$

IF $|children_p| < \delta$ THEN

$children_p := children_p \cup \{id_q\}$

SEND $YouAreMyChild(id_p)$ TO q

ELSE IF $\{id_r \in children_p | id_r > id_q\} \neq \emptyset$ THEN

$let id_s \in \{id_r \in children_p s.t. id_r > id_q\}$

SEND $Exists(id_q)$ TO s

ELSE

$let id_s \in children_p$

$children_p := chilren_p \setminus \{id_s\} \cup \{id_q\}$

SEND $YouAreMyChild(id_p)$ TO q

The algorithm - details (2)

Reception of $\mathit{YouAreMyChild}(id_q) \rightarrow$

$\mathit{Sanity\_check}(p)$

IF $parent_p = id_p$ AND $id_q > id_p$ THEN

$parent_p := id_q$

Outline

Definition of stability

$\mathcal{L}$

A configuration $C \in \mathcal{L} iff, \forall p \in P$:

(1 - unique path from any process to Max)

$$p \neq Max \Rightarrow \exists p_1, \ldots p_n \in P : (p = p_1) \wedge (p_n = Max)$$
$$\wedge \forall i \in \{1, \ldots n-1\} parent_{p_i} = id_{p_{i+1}} \wedge id_{p_i} \in children_{p_{i+1}}$$

(2 - heap invariant)

$$parent_p \geq id_p$$

(3 - I am the child of a process)

$$children_p = \{q \in P \mid parent_q = id_p\}$$

(4 - degree bound)

$$|children_p| \leq \delta$$

(5 - communications)

every $c_{p \to q} \in \mathcal{C}$ is empty or contains $Neighbor?(p)$ messages

Sketch of proof

1. Closure (*Once in $\mathcal{L}$, we remain in $\mathcal{L}$*)
2. Correctness (*In $\mathcal{L}$, the algorithm respects its specifications*)
3. Convergence (*From anywhere, we enter $\mathcal{L}$ in a finite time*)

Sketch of proof

1. Closure (*Once in $\mathcal{L}$, we remain in $\mathcal{L}$*)
2. Correctness (*In $\mathcal{L}$, the algorithm respects its specifications*)
3. Convergence (*From anywhere, we enter $\mathcal{L}$ in a finite time*)

Sketch of proof

1. Closure (*Once in $\mathcal{L}$, we remain in $\mathcal{L}$*)
2. Correctness (*In $\mathcal{L}$, the algorithm respects its specifications*)
3. Convergence (*From anywhere, we enter $\mathcal{L}$ in a finite time*)

Outline

Experimental settings

- Platform
  - Grid Explorer platform
  - 150 bi-Opteron
  - Gigabit Ethernet
- Deployment
  - up to 100 processes per node
  - logger gathering information based on local history
- Implementation
  - adapting timeout for spontaneous rule
  - RD daemon (global id_{Max}) communicating by multicast
  - failure detector service based on heartbeat
- Set-up
  - 750 to 10050 processes
  - $\delta = \{3, 4, 5\}$
  - initial configuration : disconnected network

Experimental results

Two phases :

- First, processes form trees :
  - optimal depth (logarithmic in the number of processes)
  - more efficient by increasing the degree
- Second, trees merge :
  - depth increases linearly in number of tree merging
  - number of merging linear in the number of nodes
- Stabilization time : 10000 processes $\rightarrow$ 100 seconds

Outline

Conlusion and future works

- A model for large scale self-stabilization
  - neighbors list
  - resource discovery service
  - failure detector
- Illustration
  - spanning tree
  - degree bounded
  - formal proof of convergence
  - prototype implementation and experimentation
- Open problems
  - no formal evaluation of the stabilization time
  - other problems ?
  - other topologies ?
  - realistics assumptions on the resource dicovery service