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 $\rightarrow$ dynamic neighborhood
  - failure detection service $\rightarrow$ 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 $\rightarrow$ dynamic neighborhood
  - failure detection service $\rightarrow$ 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 $\rightarrow$ dynamic neighborhood
  - failure detection service $\rightarrow$ crash awareness
Outline

1. Model
2. An example: spanning Tree Algorithm
3. Stabilization
4. Experimental Measurements
5. Conclusion
Outline

1. Model
2. An example: spanning Tree Algorithm
3. Stabilization
4. Experimental Measurements
5. Conclusion
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
- \(\mathcal{P} \subseteq \mathcal{I}\), the set of correct processes
- \(\mathcal{C} = \{c_{a \rightarrow b} | \forall a, b \in \mathcal{I}^2\}\), 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 \(c \in \mathcal{C}\)
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 $\mathcal{L}$.
Resource discovery

- Oracle providing identifiers in I:
  - assumes an id eventually returned
  - example: enumerates 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 : I → boolean
  - after a finite time return true iff the id ∉ 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

1. Model
2. An example: spanning Tree Algorithm
3. Stabilization
4. Experimental Measurements
5. Conclusion
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**:
  - $id_p$
  - $\delta$

- **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$_q$ ∈ children$_p$ ∪ \{parent$_p$\} \ \{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$_q$ ∈ children$_p$/id$_q$ < id$_p$\}

- **Suspect**(id$_p$)

- **Detect_failures**($p$):
  IF parent$_p$ ≠ id$_p$ ∧ Suspect(parent$_p$) THEN parent$_p$ = id$_p$
  \forall id$_q$ ∈ children$_p$ IF Suspect(id$_q$) THEN children$_p$ = children$_p$ \ {id$_q$}

- **RD_Get**()
The algorithm - details (2)

True →

\[\text{Sanity\_check}(p); \text{Detect\_failures}(p)\]
\[\forall id_q \in \text{Neighborhood}(p) \text{ SEND Neighbor?}(id_p) \text{ TO } q\]
\[\text{IF } \text{parent}_p = id_p \text{ THEN}\]
\[\quad id_q := \text{RD\_Get}()\]
\[\quad \text{IF } id_q > id_p \text{ THEN SEND Exists}(id_p) \text{ TO } q\]
The algorithm - details (2)

Reception of \textit{Neighbor}(id_q) \rightarrow

\texttt{Sanity\_check}(p)

\textbf{IF} id_p < id_q \textbf{THEN}

\texttt{IF} parent_p = id_p \texttt{THEN} parent_p := id_q

\textbf{ELSE IF} id_q \notin \textit{children}_p \textbf{THEN}

\texttt{IF} |children_p| < \delta \texttt{\ OR} (|children_p| = \delta \texttt{\ AND} \exists id_r|id_r < id_q) \textbf{THEN}

\texttt{children}_p := children_p \setminus id_r \cup \{id_q\}

\textbf{ELSE IF} id_p \neq id_q \textbf{THEN}

\texttt{SEND NotNeighbor(id_p) TO q}
The algorithm - details (2)

Reception of $\text{NotNeighbor}(id_q) \rightarrow$

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

IF $\text{parent}_p = id_q$ THEN $\text{parent}_p : id_p$

$\text{children}_p = \text{children}_p \setminus \{id_q\}$
The algorithm - details (2)

Reception of \( \text{Exists}(id_q) \) →

\[ \text{Sanity\_check}(p) \]

IF \(|\text{children}_p| < \delta\) THEN

\[ \text{children}_p := \text{children}_p \cup \{id_q\} \]

SEND \( \text{YouAreMyChild}(id_p) \) TO \( q \)

ELSE IF \( \{id_r \in \text{children}_p | id_r > id_q\} \neq \emptyset \) THEN

let \( id_s \in \{id_r \in \text{children}_p| \text{s.t.} \ id_r > id_q\} \)

SEND \( \text{Exists}(id_q) \) TO \( s \)

ELSE

let \( id_s \in \text{children}_p \)

\[ \text{children}_p := \text{children}_p \setminus \{id_s\} \cup \{id_q\} \]

SEND \( \text{YouAreMyChild}(id_p) \) TO \( q \)
The algorithm - details (2)

**Reception of** \textit{YouAreMyChild}(id_q) \rightarrow
\begin{align*}
\textit{Sanity\_check}(p) \\
\text{IF } parent_p = id_p \text{ AND } id_q > id_p \text{ THEN} \\
parent_p := id_q
\end{align*}
Outline

1. Model
2. An example: spanning Tree Algorithm
3. Stabilization
4. Experimental Measurements
5. Conclusion
Definition of stability

\[ L \]

A configuration \( C \in Liff \), \( \forall p \in P : \)

1. **unique path from any process to Max**
   \[
   p \neq \text{Max} \Rightarrow \exists p_1, \ldots, p_n \in P : (p = p_1) \land (p_n = \text{Max}) \\
   \land \forall i \in \{1, \ldots, n - 1\} parent_{p_i} = id_{p_{i+1}} \land id_{p_i} \in children_{p_{i+1}}
   \]

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**
   \[
   \text{every} c_{p \rightarrow q} \in C \text{ is empty or contains Neighbor?}(p) \text{ messages}
   \]
Sketch of proof

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

1. Closure (Once in $L$, we remain in $L$)
2. Correctness (In $L$, the algorithm respects its specifications)
3. Convergence (From anywhere, we enter $L$ 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

1. Model
2. An example: spanning Tree Algorithm
3. Stabilization
4. Experimental Measurements
5. Conclusion
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 → 100 seconds
Outline

1. Model
2. An example: spanning Tree Algorithm
3. Stabilization
4. Experimental Measurements
5. Conclusion
Conclusion 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 discovery service