Realizing Self-stabilization for Autonomic Control Systems

Reuven Yagel, WTEC\Rafael
Advised by: Prof. Shlomi Dolev, Ben-Gurion University of the Negev
Outline

• Autonomic computing systems
• Self-stabilization
• Realizing Self-stabilization
  – Self-stabilization in Operating Systems
• Applications for Control Systems
Motivation

• Growing use of autonomous and remote computing systems (e.g. RFID, sensor networks)
• Human management is too expensive, risky or just unavailable
• The combination and type of faults cannot be totally anticipated in on-going systems (e.g. soft errors)

The result: unpredictable system states

Pentium HALTING problem: “… if the ESP or SP register is 1 when the PUSH instruction is executed, the processor shuts down…”
Towards Autonomic Computer

• Fault tolerant system: continues correct operation in the presence of faults

• Duplication:
  – Replication\redundancy\diversity
  – Many known examples: ECC, DMR\TMR, RAID, TCP retransmission

• Self-stabilization!
**Beginning dump of physical memory**

Restart and set the recovery options in the system control panel or the `/CRASHDEBUG` system start option. If this message reappears, contact your system administrator or technical support group.
Self-stabilization

• A self-stabilizing system is a system that can automatically recover following the occurrence of (transient) faults

• “Self-Stabilization in Spite of Distributed Control” [Dijkstra ’74]

• Self-Stabilization [Dolev ’00, MIT Press]
Realizing Self-stabilization

- Traditionally used in distributed systems (where transient faults are frequent)
- System can be started in an arbitrary state and converge to a desired behavior
- Cannot run self-stabilizing algorithms unless hardware+OS are stabilizing - Fair composition [Dolev ’00]
- Ben-Gurion group goal: practical self-stabilizing systems:
  - Micro-processor [Dolev, Haviv ’04]
  - Operating System [Dolev, Yagel ’04]
  - Preserving Compiler [Dolev, Haviv ‘05]
  - Autonomic Recoverer [Brukan, Dolev, Kolodner ’03]
  - Distributed File System, Middleware, …
First Algorithm: Token Passing
Token Passing Cont.

1 \( P_1: \text{ do } \text{ forever} \)
2 \( \text{ if } x_1 = x_n \text{ then} \)
3 \( x_1 := (x_1 + 1) \mod (n+1) \)
4 \( P_i (i \neq 1): \text{ do } \text{ forever} \)
5 \( \text{ if } x_i \neq x_{i-1} \text{ then} \)
6 \( x_i := x_{i-1} \)
Token Passing Cont.

- Surely works when we start in
  \[ x_1 = x_2 = \ldots = x_n = 0. \]

- One processor may change a state at a time.
Token Passing: Faults

- Transient fault, soft errors, wrong CRC, unexpected temporal severe conditions, etc.
- Assigns each processor with an arbitrary state (in the range of its state space).
- For example \{3; 4; 4; 1; 0\}.
- \(p_2, p_4,\) and \(p_5\) have tokens!
- Will the system ever recover?
Token Passing: Automatic Recovery

- \( p_1 \) changes state infinitely often,
- Otherwise, let \( s_1 \) be the fixed state of \( p_1 \),
- \( p_2 \) eventually copies \( s_1 \) from \( p_1 \), then
- \( p_3 \) eventually copies \( s_1 \) from \( p_2 \), then
- \( \ldots \)
- \( p_n \) eventually copies \( s_1 \) from \( p_{n-1} \), then
- \( p_1 \) changes state.
- \( p_1 \) changes state in the order 4; 5; 0; 1; 2; 3; 4; 5; 0; \ldots
Token Passing: Automatic Recovery Cont.

• In any initial state at least one state is missing, \{ 4; 4; 1; 0; 2 \}, 3 and 5 are missing.

• Once \( p_1 \) reaches the missing state e.g., 5, all the processors must copy 5, before \( p_1 \) reads 5 from \( p_n \) and changes state to 0.
Will It Stabilize With mod (n - 2)?

Mod 3

\[
\begin{align*}
\{0,0,2,1,0\} & \quad p_1 & \quad \{1,0,2,1,0\} & \quad p_5 \\
\{1,0,2,1,1\} & \quad p_4 & \quad \{1,0,2,2,1\} & \quad p_3 \\
\{1,0,0,2,1\} & \quad p_2 & \quad \{1,1,0,2,1\} & \\
+1 & \mod 3
\end{align*}
\]
Proposed solution

• Build on well designed and well understood paradigm of self-stabilization
• Thereby achieving: trustworthiness, dependability, self-healing, automatic recovery, adaptive systems, …

• On going research
  – Berkeley: Recovery Oriented Computing
  – JHU: The Coyotos Secure Operating System
  – IBM: K42, Autonomic Computing
  – SUN: Solaris 10, Predictive Self-Healing
  – MSR: “Singularity” – managed code OS
  – Intel: Itanium 2, Machine Check Architecture
Directions for Operating Systems

• Black-box
  – Take existing (Desktop\Real-time) OS
  – Add stabilization layer
• Carefully tailoring a tiny kernel
  – Processor scheduling
  – Memory management
  – Device drivers
• Combined: Micro-kernel\Hyper-visor
Example: Memory Management Requirements

- Consistency of memory hierarchy
- Self-stabilization preservation
Example: Memory Manager

- Continuous Consistency check and establishment
- Detailed proof for self-stabilization of algorithms AND implementation
- A few hundred lines of Pentium assembly code
Applicability to Control Systems

- Insure safety conditions *eventually*
- Algorithms for basic primitives of distributed computing:
  - Data links and routing, mutual exclusion, synchronization, …
- Leader election
  - Using randomization to break symmetry
  - Prototype implementation of multiple agents computing “Best Estimate Trajectory”
Consensus

• Distributed agreement on some value's.
• One-shot vs. continuous
• Impossibility result for $n < 3m+1$ [Lamport+ '82]
• Stability of long-lived consensus [Davidovitch, Dolev, Rajsbaum '04]
  – Avoiding unwanted jumps
  – “instability” measures
Conclusion

• We show theoretical and practical ways to achieve the goal of a self-stabilizing systems
• We believe that the research community & industry (e.g. self-* and autonomic computing) should benefit from the foundation of self-stabilization

• www.cs.bgu.ac.il/~yagel/sos

Thank you