How Resilient are Distributed *f* Fault/Intrusion-Tolerant Systems?

Paulo Sousa, Nuno Ferreira Neves, and Paulo Veríssimo

Agenda

- Introduction
- Physical Model System (PSM)
- Dependability under PSM
- An Attack to the Proactive Recovery Scheme of CODEX
- Conclusions

- Sousa, P., Ferreira, N., & Veríssimo, P. (2005). How Resilient are Distributed f Fault/Intrusion-Tolerant Systems? Proceedings from DSN'05: *The 2005 International Conference on Dependable Systems and Networks.* Los Alamitos, California: IEEE Computer Society.
  - Paper and presentation slides available on author's website

Lessons to be Learned

- Asynchronous fault-tolerant distributed systems with asynchronous proactive recovery (APR) are vulnerable and prone to failures
- A new predicate is introduced to solve this problem: exhaustion-safety
  - Applicable to other types of systems (e.g., synchronous)
- Possible solutions to limitations in async. systems with proactive recovery

Introduction (1)

- How to build a fault-tolerant distributed system?
  - Step 1: Estimate the maximum possible number of node failures N_f.
  - Step 2: Build an f fault-tolerant system, such that $f > N_f$
- How to estimate N_f?

Introduction (2)

- Estimating N_f depends on:
  - Type of faults
    - Accidental faults may be predicted
    - Malicious faults are difficult to predict
  - Synchrony assumptions
    - Synchronous system: exec time may be bounded
    - Asynchronous system: exec time is unbounded
- How to estimate N_f in an asynchronous system with malicious faults?

Introduction: Focusing on Resources

- Fault and timing assumptions are an abstraction of the required resources
  - `f` fault-tolerance means `(n-f)` correct nodes are required
- Resource exhaustion: violation of a resource assumption
  - `f+1` nodes fail
- Definition: An exhaustion-failure is a failure that results from resource exhaustion
- Definition: A system is exhaustion-safe if it ensures that exhaustion-failures never happen

Agenda

- Introduction
- **Physical Model System (PSM)**
- Dependability under PSM
- An Attack to the Proactive Recovery Scheme of CODEX
- Conclusions

Physical System Model (PSM)

- Allows to formally reason about how exhaustion-safety is affected by different combinations of timing and fault assumptions
- A system execution is defined by
  - t_{start}: the RT start instant
  - t_{end}: the RT termination instant
  - $t_{exhaust}$: the RT instant when exhaustion occurs
- Definition: A system is exhaustion-safe iff $t_{end} < t_{exhaust}$, for all executions
  - A `f` fault-tolerant distributed system is exhaustion-safe if it terminates before `f+1` components fail

Exhaustion-Safe

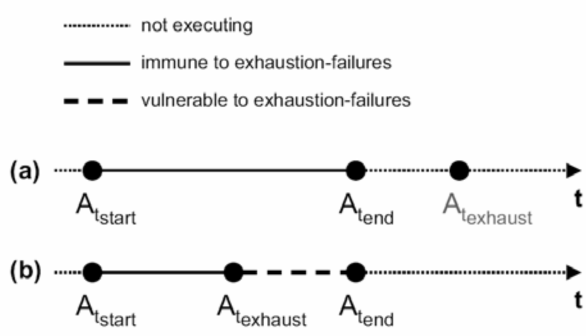

Figure 1. (a) Exhaustion-safe system; (b) non exhaustion-safe system.

Agenda

- Introduction
- Physical Model System (PSM)
- **Dependability under PSM**
- An Attack to the Proactive Recovery Scheme of CODEX
- Conclusions

Synchronous Systems under PSM

- Synchronous system properties:
  - P1 – known bound on local processing time
  - P2 – known bound on message delivery time
  - P3 – known bound on the drift rate of local clocks
- P1+P2+P3 allows to define a bound T_{end} on the execution time (if the lifespan is not unbounded)
  - But timing failures may occur, namely in a malicious environment
- Exhaustion-safe if $t_{exhaust} > t_{end}$, for all executions

Async Systems under PSM (1)

- Asynchronous system properties:
  - P1 – unbounded local processing time
  - P2 – unbounded message delivery time
  - P3 – unbounded drift rate of local clocks
- P1+P2+P3 = unbounded t_{end}
  - Immune to timing failures
  - But, how to guarantee $t_{end} < t_{exhaust}$?
- Non exhaustion-safe if $t_{exhaust}$ is bounded
  - A distributed `f` fault-tolerant async system is not exhaustion-safe

Async Systems under PSM (2)

- Real systems have a bounded $t_{exhaust}$
  - Resources degrade over time (HW failures, SW bugs, malicious attacks)
  - Accidental degradation is different from malicious degradation
    - Accidental faults occur in a random manner and can be studied
    - Malicious faults occur in the most convenient manner for the adversary
- Thus, $t_{exhaust}$ should not be bounded in async systems operating in malicious environments (e.g., Internet)

Proactive Recovery (1)

- Goal: to constantly postpone $t_{exhaust}$ through periodic rejuvenation
  - Periodic rejuvenation of OS
- A system is exhaustion-safe only if rejuvenations are always terminated before exhaustion

Proactive Recovery (2)

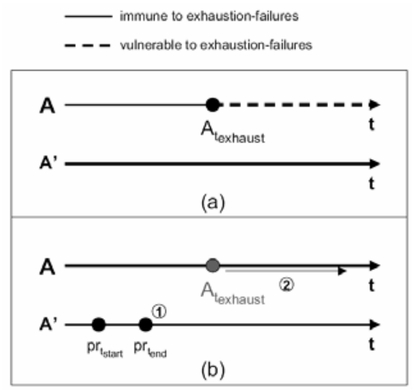

Figure 2. (a) Before proactive recovery being executed, A exhaustion-safety is in risk of being violated; (b) after the execution of proactive recovery (1), $A_{t_{exhaust}}$ is postponed (2).

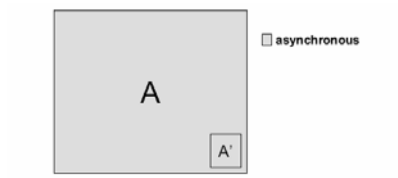

Figure 3. A system A enhanced with a proactive recovery subsystem A'. Both A and A' run asynchronously.

Asynchronous Proactive Recovery

- How to guarantee that rejuvenations always terminate before resource exhaustion?
  - Rejuvenation start instant may be delayed
  - Rejuvenation actions may be delayed
  - These delays may be enforced by a malicious adversary
- Asynchronous proactive recovery does not guarantee exhaustion-safety
  - Namely in a malicious environment

Agenda

- Introduction
- Physical Model System (PSM)
- Dependability under PSM
- **An Attack to the Proactive Recovery Scheme of CODEX**
- Conclusions

Attack to CODEX (1)

- Proactive Recovery Strategy
  - A1: $f \leq (n-1)/3$ servers compromised, $\forall t$
  - A2: mobile virus attacks can occur, leading attacker to learn `f+1` shares
  - A3: APSS triggered periodically, sufficiently often to prevent A1 from being violated
    - Key is compromised if an adversary collects sufficient shares in the interval between successive executions of the APSS

Attack to CODEX (2)

- Theory
  - Non exhaustion-safe systems are prone to failure
  - Asynchronous proactive recovery systems cannot be exhaustion-safe
- Conjecture about real systems
  - Timing assumptions (e.g., “periodicity”, “sufficiently often”) are unsustainable, leading to possible failure scenarios
  - In malicious settings, the above is not only possible, but probable

Attack to CODEX (3)

- Experiment (to confirm theory) performed by two adversaries: ADV1 and ADV2
  - Step 1: ADV1 performs a mobile virus attack against `f+1` servers
    - Slow down the clock rate of each server
  - Step 2: ADV1 temporally cuts off the links between the `f+1` servers and the rest of the system
  - N.B. – ADV1 actions are allowed behavior of the system/network
    - Simply enforce a behavior that can occur in any fault-free asynchronous system

Attack to CODEX (4)

- Experiment (to confirm theory) performed by two adversaries: ADV1 and ADV2
  - Step 3: ADV2 performs a mobile virus attack against the same `f+1` servers
    - Learns, one by one, `f+1` private key shares
    - No rejuvenation occurs in between because in step 1 clocks are made as slow as needed
  - Step 4: ADV2 discloses the private key by combining the `f+1` shares

Posible Solution to E-S in APSS

- Based on wormholes
  - Subsystems capable of providing services with good properties, otherwise not available in system
- Authors provide evidence of distributed wormholes (previous work, implemented in RTAI Linux O/S)

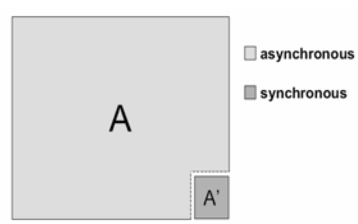

Figure 5. A system A enhanced with a proactive recovery subsystem A'. A runs asynchronously, but A' runs synchronously in the context of a secure and timely wormhole.

Conclusions (1)

- Showed that current state-of-the art leads to the construction of systems that are not exhaustion-safe and thus prone to failure
  - Sync systems are vulnerable (timing failures)
  - Async systems are vulnerable (max no. of faults + unbounded exec time)
  - Async systems with APR are vulnerable (max no. of faults + unbounded rejuvenation period)

Conclusions (2)

- Proposed new system model that opens avenues to characterize and solve the problem
  - Any system must possess the exhaustion-safe predicate