Reliability

A CPS Perspective

Results

- Average: 26.25/30  (87.5%)
Results

- **Average**: 26.25/30  (87.5%)

Three Classical Challenges

- **Establish Functional Correctness**: How to build functionally correct systems from possibly flawed components?
- **Establish Temporal Correctness**: What are the analytic foundation for robust timing guarantees in highly dynamic, time-critical software systems?
- **Establish Data Correctness**: How to ensure reliable operation in the face of unreliable data?
Three Classical Challenges

- **Establish Functional Correctness**: How to build functionally correct systems from possibly flawed components?
- **Establish Temporal Correctness**: What are the analytic foundation for robust timing guarantees in highly dynamic, time-critical software systems?
- **Establish Data Correctness**: How to ensure reliable operation in the face of unreliable data?

Rate of Innovation and Development Time Issues

- Near the turn of the 20th century products had a 20-30 year life-span before new “versions” were developed
- At present, a product is obsolete in 2-3 years
  - No time to discover and “debug” all possible problems
  - New problems introduced in new versions
  - Component reuse generates additional problems
Software: Increasingly the Primary Cause of System Failure

- Arbitrary component interactions unconstrained by physical laws of nature (algorithms can do anything)
  - Potential for high interactive complexity
- Fast error propagation (at computing device speed)
  - Potential for tight coupling
- Software that interacts with the physical world is buggy!

Typical Isolation Techniques

- Abstraction
- Separation of concerns

Abstraction

- Transport
- Network
- Link
- Physical

Separate virtual machines or protection domains

Kernel

Virtualization
Abstraction → Specialization

- Complexity
  - More levels of abstraction
  - Narrower specialization
  - More details are “abstracted away”
  - Myopic view. Less knowledge of possible adverse interactions
  - More potential for interaction or incompatibility errors

The Curse of Component Re-use

The Ariane 5 Explosion

- On June 4, 1996, the maiden flight of the European Ariane 5 launcher crashed about 40 seconds after takeoff (0.5 Billion Dollars)
- Cause of problem?
  - An inertial reference software component.
    - Not needed during flight. Should be stopped before takeoff but is allowed to operate for up to 50 additional seconds to avoid expensive restarts should countdown be interrupted
    - Component was designed for Ariane 4. Ariane 5 was a faster system. Velocity variable overflowed.
    - Overflow causes an exception that is not caught and crashes the software
Example 1: Interactive Complexity in Distributed Protocols

- Interactive complexity means:
  - Simple individually insignificant failures interact to compound into system failures, or even...
  - Sets of correctly operating components interact to produce a system failure
    - Example:
      - Shortest hop routing
      - Adaptive rate control

Example 1:
- Shortest hop routing
  - Find shorter path (fewer hops that are longer)
- Long wireless hops → poor channel quality
- Adaptive rate control
  - Reduce transmission rate to improve quality
- Reduced transmission rate
  → longer transmission range
Example 2: Correlated failure modes between “independent components”

- Localization (determining a node’s location) fails in a correlated manner with failure to synchronize clocks. Why?
  - Note: None of the two components uses the other

- Answer: communication problems. Both subsystems rely on distributed protocols

Poor performance is Compounded by two Correlated failures
Example 3:
More on hidden interactions

- Magnetic tracking system operates perfectly in calm weather but fails under strong wind conditions. Why?
  - Wind should not change magnetic sensor reading

Explanations:
- Wind caused node antenna to vibrate
- Moving (metal) antenna caused a lot of noise on the magnetic sensor
- Noise filter adapted noise threshold to remove background noise (and in this case the signal too)
Example 4: Three Mile Island Nuclear Reactor Failure

- False alarm of minor secondary system coolant leakage through seal
- Stop secondary coolant flow and turbine
- Heat exchange stops between primary and secondary cooling systems. Primary overheats.
- Valve failure indicator light turns on but is occluded by repair tag on another device
- Core overheating triggers emergency shutdown
- Coolant pressure relief valve opens to reduce pressure
- Pressure drops. Valve is stuck open. Coolant boils off. Core temperature rises. Reaction resumes.
- Core is flooded with water
- Water at very high temperature oxidizes metal fuel rod coating (rusting)
- Hydrogen is released eventually leading to explosion
- Core temperature and pressure continue to build up
- Open emergency feed-water pumps from emergency tank to cool coolant

Ensuring Software Correctness

- The physical world has no “reset” button
  - When failures occur, they can be costly!
- Must reduce:
  - Interactive complexity
    - Unexpected interactions between seemingly correct components
  - Coupling
    - Fast propagation of effects of failure to other system components
Designing Complex Systems
(Example: Air-traffic control)

- Reduce interactive complexity
  - Air traffic is restricted to non-intersecting “corridors” that separate flight paths in the sky

- Reduce coupling
  - Separate aircraft by a substantial distance to reduce cascaded failure effects (think: multiple-car pile-ups in freeway accidents)

Interaction Examples

- Function calls
- Resource sharing
  - One module crashes → overwrites memory of another → second “unrelated” module crashes (analogy to physical proximity and correlated damage)
  - One module is overloaded → another starves
- Timing and synchronization constraints
  - Precedence constraints (one module must execute before another)
  - Exclusion constraints (cannot operate at the same time)
- Assumptions
  - I thought you submitted our paper?
  - No, I thought you did?
The Performance/Robustness Trade-off

**Performance:** Exploring the edge of stability

**Robustness:** Guaranteeing delivery in the face of adverse conditions

Interactive Complexity in Networked Systems

- Attaining high performance requires:
  - More complex interactions
  - More dependencies
  - Deeper cascading failures
  - Lower robustness

- Cascading failure on “high-performance” road
- Non-cascading failure on side-street
**Challenge**

Investigate performance/robustness trade-off
- Offers more graceful degradation with loss

Investigate the role of interactions and coupling
- Model propagation of local failures globally

**Candidate Approach: Formal Methods**

- Express safety properties (e.g., task A will never miss its deadline)
- Prove that safety properties hold
  - If proof fails, counter example is presented (a sequence of events that leads to failure)
- Problem:
  - Proofs require axioms. Axioms may make incorrect assumptions (e.g., circular sensing range)
  - Interactions must be explicitly modeled. Failure to model interactions (e.g., between wind and magnetic sensor) may overlook some failure modes.
Living with Buggy Systems

- If errors cannot be avoided (even using formal methods), we must design systems to tolerate them
  - Architectures for “living with bugs”
  - Fast diagnosis and recovery
  - Issues
    - Problem must be observable (or else cannot diagnose)
    - Observation must be in time so that recovery is possible (observing that you forgot your parachute after you jump will not help you)
    - Systems with highly auto-correlated state on long time-scales will likely take long to recover

Simplicity to Conquer Complexity

Lui Sha

- Elements of a good design
  - Simple safety core
  - Complex enhanced mission functionality
  - Formal proof of core correctness
  - Well formed dependency (core may use but will not depend on any other components)
Diagnosis:
A Development-Time Data Mining Example

- Run system multiple times
- Log all observable interactions (messages exchanged, resources allocated, etc)
- Label execution as “correct” (no observable problems) or “incorrect” (problems observed)
- Separate logs into “good” data set and “bad” data set
- Look for sequences of events in the “good” pile but not the “bad” pile

Three Classical Challenges

- **Establish Functional Correctness**: How to build functionally correct systems from possibly flawed components?
- **Establish Temporal Correctness**: What are the analytic foundation for robust timing guarantees in highly dynamic, time-critical software systems?
- **Establish Data Correctness**: How to ensure reliable operation in the face of unreliable data?
Three Classical Challenges

- **Establish Functional Correctness**: How to build functionally correct systems from possibly flawed components?
- **Establish Temporal Correctness**: What are the analytic foundation for robust timing guarantees in highly dynamic, time-critical software systems?
- **Establish Data Correctness**: How to ensure reliable operation in the face of unreliable data?

Goal: Analyze Distributed Highly Coupled Systems

Complex timing behavior: Everything depends on everything
Three Classical Challenges

- **Establish Functional Correctness:** How to build functionally correct systems from possibly flawed components?
- **Establish Temporal Correctness:** What are the analytic foundation for robust timing guarantees in highly dynamic, time-critical software systems?
- **Establish Data Correctness:** How to ensure reliable operation in the face of unreliable data?
Three Classical Challenges

- **Establish Functional Correctness:** How to build functionally correct systems from possibly flawed components?
- **Establish Temporal Correctness:** What are the analytic foundation for robust timing guarantees in highly dynamic, time-critical software systems?
- **Establish Data Correctness:** How to ensure reliable operation in the face of unreliable data?

Crowd-sensing in CPS Applications
Assessing Reliability of Unvetted Data Feeds
Constructing Reliable State from Unreliable Observers

- Define $a_i$ as:
  - $P$ (source $i$, makes an original observation | it is true)
- Define $b_i$ as:
  - $P$ (source $i$, makes an original observation | it is false)
- What are the source reliability parameters that maximize the probability of received observations?

A Maximum Likelihood Estimation Problem

$$P(SC|\theta) = \sum_z P(SC, z|\theta)$$

$$\theta_{n+1} = \arg\max_\theta \left\{ E_{z|SC, \theta_n} \{ lnP(SC, z|\theta) \} \right\}$$

The expectation maximization algorithm computes:

- The log likelihood function, $lnP(SC, z|\theta)$
- The expectation step, $Q_\theta = E_{z|SC, \theta_n} \{ lnP(SC, z|\theta) \}$
- The maximization step, $\theta_{n+1} = \arg\max_\theta \{ Q_\theta \}$
Formulation:
A Maximum Likelihood Estimation Problem

- Maximize log-likelihood by appropriate selection of truth values for claims:

Log-likelihood Function of EM Scheme:

$$l_{em}(x; \theta) = \sum_{j=1}^{N} \left\{ z_j \times \left[ \sum_{i=1}^{M} \left( S_i C_j \log a_i + (1 - S_i C_j) \log (1 - a_i) + \log d \right) \right] \right\} + (1 - z_j) \times \left[ \sum_{i=1}^{M} \left( S_i C_j \log b_i + (1 - S_i C_j) \log (1 - b_i) + \log (1 - d) \right) \right]$$

where $z_j = 1$ when measured variable $j$ is true and 0 otherwise

---

Expectation Maximization

$L(\theta; X, Z) = p(X, Z|\theta)$

$$- \sum_{j=1}^{N} \sum_{i=1}^{M} q_i S_i C_j (1 - a_i) \log a_i + (1 - S_i C_j) \log (1 - a_i) \times d \times z_j + \sum_{i=1}^{M} q_i S_i C_j (1 - b_i) \log b_i + (1 - S_i C_j) \log (1 - b_i) \times (1 - d) \times (1 - z_j)$$

Expectation Step (E-Step)

$$Q(\theta^{(t)}) = E_{Z|X, \theta^{(t)}} \log L(\theta; X, Z)$$

$$- \sum_{j=1}^{N} \sum_{i=1}^{M} q_i S_i C_j (1 - a_i) \log a_i + (1 - S_i C_j) \log (1 - a_i) + \log d$$

$$+ p(z_j = 1|X, \theta^{(t)}) \times \sum_{i=1}^{M} q_i S_i C_j (1 - b_i) \log b_i + (1 - S_i C_j) \log (1 - b_i) \log (1 - d)$$

Maximization Step (M-Step)

$$a_i^{(t+1)} = a_i^* = \frac{\sum_{j=1}^{N} q_i S_i C_j Z(t, j)}{\sum_{j=1}^{N} q_i S_i C_j Z(t, j, j)}$$

$$b_i^{(t+1)} = b_i^* = \frac{K_i}{N} \sum_{j=1}^{N} a_i^{(t)} Z(t, j, j)$$

Iterate

$$\hat{e}^{(t+1)} = \frac{\sum_{j=1}^{N} q_i S_i C_j Z(t, j, j)}{N}$$
Derivation of Confidence
Using the Cramer-Rao Bound

Estimation and Statistic Background
Fisher information is defined as
\[ I(\theta) = E_x \left[ \varphi(x;\theta)^T \varphi(x;\theta) \right] \]
Score vector \( \varphi(x;\theta) \) for a \( k \times 1 \) estimation vector \( \theta = [\theta_1, \theta_2, \ldots, \theta_k]^T \)
\[ \varphi(x;\theta) = \left[ \frac{\partial l(x;\theta)}{\partial \theta_1}, \frac{\partial l(x;\theta)}{\partial \theta_2}, \ldots, \frac{\partial l(x;\theta)}{\partial \theta_k} \right]^T \]
Fisher Information Matrix can be rewritten as (under regularity condition of EM):
\[ (I(\theta))_{i,j} = -E_x \left[ \frac{\partial^2 l(x;\theta)}{\partial \theta_i \partial \theta_j} \right] \]
Cramer-Rao Bound (CRB) is defined as the inverse of Fisher information
\[ CRB = I^{-1}(\theta) \]

Event Monitoring

- Define \( \alpha_i \) as:
  - \( P \) (source \( i \) makes an original observation | it is true)
- Define \( \beta_i \) as:
  - \( P \) (source \( i \) makes an original observation | it is false)
- What are the source reliability parameters that maximize the probability of received observations?

\[ P(SC|\theta) = \sum_z P(SC, z|\theta) \]
Event Monitoring

- Define $\alpha_i$ as:
  - $P$ (source $i$ makes an original observation | it is true)
- Define $\beta_i$ as:
  - $P$ (source $i$ makes an original observation | it is false)
- What are the source reliability parameters that maximize the probability of received observations?

Reconstructing Event Timelines

A Twitter Example

- Define $\alpha_i$ as:
  - $P$ (source $i$ makes an original observation | it is true)
- Define $\beta_i$ as:
  - $P$ (source $i$ makes an original observation | it is false)
- What are the source reliability parameters that maximize the probability of received observations?

$P(SC|\theta) = \sum_{z} P(SC, z|\theta)$
Tool Available Online

Link: apollofactfinder.net

Book: Social Sensing: Building Reliable Systems on Unreliable Data
Dong Wang, Tarek Abdelzaher, Lance Kaplan, Morgan Kaufmann, 1st Edition, April, 2015