Building Consensus: Foundations of Monitoring Ultra-Reliable Systems

Lee Pike leepike@galois.com
Galois, Inc.

Alwyn Goodloe alwyn.goodloe@nianet.org
National Institute of Aerospace
The Problem: Motivation

Space Shuttle

• In 2008, a pre-launch failure of STS-124 was reported in the Space Shuttle’s data processing system.

• Components:
  - FA 2: the flight-aft mux/demux card
  - GPC n: general-purpose computers n

• The incident:
  1. A diode fails on FA 2.
  2. GPC 4 receives bad data from FA 2; in the data comparisons with GPC 1-3, it is voted out.
  3. Then similarly for GPC 2.
  4. GPC 3 also determined to be faulty.
  5. With only one GPC remaining, the system was powered-down.

• Described as a “non-universal I/O error”
Characterizing the Systems

The systems we focus on must be ultra-reliable, and so demand catastrophic-failure rates of $\geq 10^{-9}$ per hour of operation.

- They’re fault-tolerant, meaning they
- have replicated hardware & distributed architectures
- and have fault-management SW,
- and are hard real-time.
Previous research on monitors mostly focuses on systems lacking **one or more** characteristics of ultra-reliable systems.

- Much focus on *inline monitors* for software, particularly Java programs, e.g.,
  - Run-time Monitoring and Checking (MaC) - Insup Lee et al.
  - Monitoring-Oriented Programming (MOP) - Rosu et al.
- Efforts to compile specifications to **efficient** inline monitors.
- Specification-logics aim to capture properties about program traces.
Previous Efforts

A few efforts have touched on aspects of safety-critical embedded systems. Representative efforts include:

- MOP extensions to monitor distributed programs using a past-time modal logic.\(^1\)
- BusMOP: synthesizing high-level specs onto FPGAs for zero-overhead bus monitoring.\(^2\)
- Logics for monitoring real-time systems (particularly distributed Java programs).\(^3\)

\(^1\)[Sen, Vardhan, Agha, Rosu. Efficient Decentralized Monitoring of Safety in Distributed Systems, ICSE’04.]
\(^2\)[Pellizzoni, Meredith, Caccamo, Rosu. Hardware Runtime Monitoring for Dependable COTS-based Real-Time Embedded Systems, RTSS’08.]
\(^3\)[Mok and Liu. Efficient Run-Time Monitoring of Timing Constraints. RTAS’97.]
Research Agenda

• Our research aims at monitoring for faults. Specifically, we want to know when a fault is systematic or beyond the system's fault model.

• We focus on monitor synthesis for checking consensus in distributed hard real-time systems.

• So what's new?
  - Our approach marries runtime monitoring with fault detection.
  - We propose that HW & SW cannot be separated when considering reliability.
  - We focus on simple consensus properties.
Outline

1. Context setting: previous work
2. Consensus properties
3. Monitor requirements
4. Conclusions
Consensus Properties

• We propose to monitor for consensus in distributed systems.

• What faults can be couched in terms of consensus?
  1. Fault-model violations
  2. Point-to-point error-checking
  3. Timing violations
Consensus Properties: Consensus

Monitoring fault-model violations

- A **maximum fault assumption** (MFA) states the maximum number of each kind of fault a system designed to withstand.
- An MFA along with the fault-arrival rate gives you its **hypothesized reliability**.
- Too often **hypothesized reliability < actual reliability**:
  - Design errors (i.e., **systematic faults**) cause the actual MFA to be a subset of the hypothesized MFA.
  - Designers **underestimate the MFA required** to achieve the desired reliability. The Shuttle incident arguably resulted from an underestimated MFA.
Consensus Properties: Consensus

• A monitor can observe consensus (or the lack thereof) between distributed components.
• This principally means observing classes of asymmetric or Byzantine faults (including omissive faults).
• It appears that Byzantine faults are also the most “malicious” and least accounted-for faults.
• Example: non-universal I/O error in the Shuttle!
• Monitors are bound by the “laws” of distributed-system observation (given real-time clocks). This means there's some probability of false-positives and false-negatives.

Example:
Consensus Properties: CRCs

Monitoring point-to-point error-checking

• Point-to-point error-checking provides evidence to a receiver that a message got corrupted in transit.
• Cyclic redundant checks (CRCs) are standard practice for catching point-to-point communication errors in embedded systems.
• They can catch both burst errors and random bit-errors.
Consensus Properties: CRCs

• Reliability figures for distributed embedded systems depend on the error-checking reliability of CRCs...

• But reliability figures may be overly-optimistic:
  “...The use of CRCs as a mechanism to provide ultra-dependable system operation (10^{-9} failures/hour) is questionable in many cases. The main problem is that network inter-stages can exhibit arbitrary faults, accidentally forging valid CRC check sequences.”

1[Paulitsch, Morris, Hall, Driscoll, Koopman, & Latronico. Coverage and the Use of Cyclic Redundancy Codes in Ultra-Dependable Systems, DSN’05.]
Consensus Properties: CRCs

For example, consider the case of “Schrödoinger’s CRCs”:

<table>
<thead>
<tr>
<th>11-Bit Message</th>
<th>USB-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Receiver A</td>
<td>1 1 1 1 1 1 0 1 1 0 1</td>
</tr>
<tr>
<td>Transmitter</td>
<td>1 1 1 1 1 1 0 1 1 0 ½</td>
</tr>
<tr>
<td>Receiver B</td>
<td>1 1 1 1 1 1 0 1 1 0 0</td>
</tr>
</tbody>
</table>

- (USB-5 has a Hamming Distance of 3 for 11-bit data.)
- No good data exists on the real-world probability of Schrödoinger’s CRCs.
- Probably more likely than commonly believed.

¹[Driscoll, Hall, Sivencrona, & Zumsteg. Byzantine Fault Tolerance, from Theory to Reality, SAFECOMP’03.]
Consensus Properties: Timing

Violated timing assumptions

Hard realtime systems have timeliness guarantees, provided system **timing assumptions** hold.

- The timing assumptions are constraints on clock drift, skew, message delays, resynchronization, etc.
- Constraints **cannot** be monitored directly.
- (A monitor has no more access to real-time than the what's monitored.)
Consensus Properties: Timing

- Constraints talk about real-time (i.e., wall-clock time).
- For example: here’s a clock drift-rate constraint:

\[
\left[(1 - \rho) \cdot (t_1 - t_2)\right] \leq C(t_1) - C(t_2) \leq \left[(1 + \rho) \cdot (t_1 - t_2)\right]
\]

- But violations of constraints will manifest themselves as systematic faults (i.e., greater than the expected fault-arrival rates).
- And faults are likely to be slightly-out-of-spec timing errors.
- Challenge: determining when a fault is frequent enough to be a systematic fault.
- Techniques for probabilistic runtime checking in soft real-time systems are applicable.¹

¹[Sammapun, Lee, Sokolsky, Regeher. Statistical runtime checking of probabilistic properties, RTV'07.]
Architectural Considerations

• What are monitors:
  - Inputs are local state projections.
  - Data are fault-arrive probabilities and state-collection times.
  - State is occurrence frequencies.
  - Outputs are consensus violations.

• Where does the monitor “go”?
  - Two architectural approaches:
    • Distributed: monitors at the distributed nodes, and interchange “consensus data”.
    • Central: nodes send “consensus data” to a central monitor.
  - Resulting in various reliability/cost tradeoffs.
  - Want to be able to synthesize multiple architectures.
Monitor Architecture Requirements

• What general requirements are there for monitor architectures?

• We propose three requirements covering
  - Functionality
  - Schedulability
  - Reliability

New
Monitor Architecture Requirements

- **Functionality**: the monitor does not change the functionality of the system under observation (SUO), unless the SUO violates its specification.
  - **Unintentional**: safe-guards must be in place to ensure that monitor faults do not affect the SUO’s functionality.
  - **Intentional**: the monitor must signal a reset, etc. to the SUO only if the SUO has (probably) violated its specification.

- **Schedulability**: the monitor architecture does not cause the SUO to violate its hard real-time guarantees.

- **Reliability**: the reliability of the SUO in the context of the monitor architecture is greater or equal to the reliability of the SUO alone. A monitor might reduce the SUO’s reliability for some class of faults of (improbable) faults and yet increase the system’s overall reliability.
Synthesis

• In other monitoring work, the synthesis challenge is
  - Synthesizing efficient monitors from expressive high-level specifications.
  - Inlining the monitors into the system.

• In ours, the challenge is to
  - Synthesize multiple architectures and ensure noninterference with the observed system.
  - Synthesize reliability data (to probabilistically distinguish systematic and random faults).
  - Synthesize temporal constraints on monitoring.
Anticipated Developer Workflow

In our context, the system designer

- Instruments processes to make “consensus data” available to the monitor (e.g., memory access).
- Provides random fault-arrival probabilities.
- Defines a monitor architecture.
- Play a game:
  - Do you assume consistency at this point in the algorithm/architecture?
  - Then assert consensus.
- Orthogonal to any fault-tolerance in the system.
Conclusions: Comments on the Approach

As compared to other monitoring frameworks...

Benefits:

- **Thesis**: consensus violations characterize a simple but broad class of faults.
  - Consensus violations characterize recent failures.
  - Consensus is hard and the assumptions are often wrong.
  - Many SW faults are about coordination and fault-tolerance rather than the core GN&C algorithms.

- Takes a unifying view of HW and SW.
  - Reliability is a function of (1) systematic and (2) random faults.
  - Thus, we take a system-level viewpoint of monitoring.
Conclusions: Comments on the Approach

As compared to other monitoring frameworks...

Challenges:
- In ad-hoc systems, which state-projections should be in agreement at which times?
- Synthesizing monitoring architectures.
- Are false-positive/negative observations acceptable (for ultra-reliable systems)?
- Is the $\delta$-increase in reliability sufficient to warrant monitoring?
Conclusions: Summary

- Ultra-reliable systems may benefit from runtime monitoring, but new approaches are needed.
- Important classes of faults can be couched in terms of consensus.
- The synthesis problem for these monitors include architectural integration and including hypothesized fault-arrival rates.
- Our hope is that “cheap and easy” consensus monitors encourage better design practices.
Conclusions

More details:

- Extended abstract accepted in the *Software Health Management Workshop* (SHM’09).
- Submitted: paper on our real-time test-bed and automated-test framework.
- In preparation: technical report survey & foundations of monitoring real-time distributed systems.
- leepike@galois.com and alwyn.goodloe@nianet.org