Dependability for Computer Systems meets Data Analytics

Saurabh Bagchi
School of Electrical and Computer Engineering
Department of Computer Science
Purdue University

Presentation available at: engineering.purdue.edu/dcs1
Roadmap

- Dependability basics
  - System design principles
  - Terminology and basic approaches
  - Case studies
- Debugging in the large: Large-scale distributed applications
  - Using metric mining (DSN 12, SRDS 13)
  - Scale-dependent bugs (HPDC 11, HPDC 13)
- Debugging in the small: Embedded and mobile platforms
  - Record and replay using hardware-software (Sensys 11)
  - Record and replay using software only (IPSN 15)

What is Dependable Computing?

- Dependability: Property that the computer system meets its specification despite the presence of faults
  - Faults can be due to natural causes (bugs, defects in hardware), or
  - Maliciously induced (attacks from external or internal sources)
- Terminology
  - **Failure**: Deviation of the delivered service from compliance with the specification
  - **Error**: Part of the system state that has been damaged by the fault and, if uncorrected, can lead to a failure.
  - **Fault**: The adjudged or hypothesized cause of an error
Two Facets of Dependable Computing

• Mercedes version
  – High hardware or software development costs
  – High power consumption
  – High space overhead
  – Example: Boeing 777 fly-by-wire (FBW) system, which used triple modular redundancy for all hardware resources
  – Example: AT&T’s ESS telecommunication switch which had a requirement of downtime < 2 minutes/year

• Commodity systems
  – Cannot have too high development costs
  – Cannot impede performance of system significantly
  – Cannot take recourse to too high redundancy
  – Example: ECC, Parity, RAID

How do We Achieve the Objectives?

- Applications
  - Application program interface (API)
  - Middleware

- Reliable communications

- Operating system
  - System network
  - Processing elements
  - Memory
  - Storage system

- Hardware
- Checkpointing and rollback, application replication, software, voting (fault masking), Process pairs, robust data structures, recovery blocks, N-version programming,
- CRC on messages, acknowledgment, watchdogs, heartbeats, consistency protocols
- Memory management, detection of process failures, hooks to support software fault tolerance for application
- Error correcting codes, N_of_M and standby redundancy, voting, watchdog timers, reliable storage (RAID, mirrored disks)
Fault Cycle & Dependability Measures

Reliability:
a measure of the continuous delivery of service; 
\( R(t) \) is the probability that the system survives (does not fail) throughout \([0, t]\);
expected value: \( MTTF(Mean\ Time\ To\ Failure) \)

Maintainability:
a measure of the service interruption; 
\( M(t) \) is the probability that the system will be repaired within a time less than \( t \);
expected value: \( MTTR \) (Mean Time To Repair)

Availability:
a measure of the service delivery with respect to the alternation of the delivery and interruptions 
\( A(t) \) is the probability that the system delivers a proper (conforming to specification) service at a given time \( t \).
expected value: \( EA = \frac{MTTF}{MTTF + MTTR + DL} \)

Safety:
a measure of the time to catastrophic failure 
\( S(t) \) is the probability that no catastrophic failures occur during \([0, t]\); 
expected value: \( MTTCF(Mean\ Time\ To\ Catastrophic\ Failure) \)

Ways to Increase System Availability

• To increase availability:
  1. Increase MTTF
  2. Decrease MTTR
  3. Decrease DL
Roadmap

• Dependability basics
  – System design principles
  – Terminology and basic approaches
  Case studies
• Overview of research projects
• Automatic problem diagnosis
  – Using metric mining (SRDS 13, HPDC 13)
  – Anomaly detection in data-dependent systems (PACT 15)

Cautionary Tales

Medical Device Failures
• 1990-2000: 600,000 cardiac devices recalled. 41% of recalls due to software issues
• 2008-12: 15% of all the medical device recalls (Class I, II & III) due to software
• Formal model checking has been proposed in research but is not widely deployed in practice

Space Launch Failures
• 1996: European Space Agency Ariane 5 satellite launcher crashed 37 seconds after launch
• 1999: Mars Polar Lander shut down its main engines and crashed before reaching the planet surface
Cautionary Tales

Roadmap

• Dependability basics
  – System design principles
  – Terminology and basic approaches
  – Case studies

Debugging in the large: Large-scale distributed applications
  – Using metric mining (DSN 12, SRDS 13)
  – Scale-dependent bugs (HPDC 11, HPDC 13)

• Debugging in the small: Embedded and mobile platforms
  – Record and replay using hardware-software (Sensys 11)
  – Record and replay using software only (IPSN 15)
**Debugging in the Large: Large-Scale Distributed Applications**

- **Goal:** Provide highly available applications (e.g., web service) in distributed environment
  - Perform failure prediction
  - Perform bug localization
- **Challenges in today’s distributed systems**
  - Large number of entities and large amount of data
  - Interactions between entities causes error propagation
  - High throughput or low latency requirements for the application

---

**Metric-based Bug Localization**

- **Application**
  - Requests rate, transactions, DB reads/writes, etc.
- **Middleware**
  - Virtual machines and containers statistics
- **Operating System**
  - CPU, memory, I/O, network statistics
- **Hardware**
  - CPU performance counters

*How can we use these metrics to localize the root cause of problems?*
Metric-based Bug Localization: Solution Approach

- Look for abnormal time patterns [Ozonat-DSN08, Gao-ICDCS09, Sharma-DSN13, ...]
- Pinpoint code regions that are correlated with these abnormal patterns

Bugs Cause Metric Correlations to Break

- Hadoop DFS file-descriptor leak in version 0.17 (2008)
- Correlations are different when the bug manifests itself:
  - Metrics: open file descriptors, characters written to disk

---

Approach Overview

Source Code

Modeling Scale-dependent Behavior

Is there a bug in one of the production runs?
Modeling Scale-dependent Behavior

Accounting for scale makes trends clear, errors at large scales obvious

Training runs  Production runs

# OF TIMES LOOP EXECUTES

Difficulty in Detecting Scale-Dependent Bugs

- What are they?
  - Bugs that arise often in large-scale runs while staying invisible in small-scale ones
  - Examples? Race condition, integer overflow, etc.
  - Severe impact on distributed systems: performance degradation, silent data corruption

- Difficult to detect, let alone localize
  - Large scale computing environment not available to developers
  - Large scale data not available at development time

When Statistical Debugging Meets Scale-dependent Bugs

• To address scale-dependent bugs with previous methods
  – Either be very restricted in your feature selection
  – Or have access to a bug-free run on the large-scale system

• Our method solved the problems of previous methods
  – Capable of modeling scaling behavior of distributed programs
  – Only need access to bug-free runs from small-scale systems

Roadmap

• Dependability basics
  – System design principles
  – Terminology and basic approaches
  – Case studies

• Debugging in the large: Large-scale distributed applications
  – Using metric mining (DSN 12, SRDS 13)
  – Scale-dependent bugs (HPDC 11, HPDC 13)

Debugging in the small: Embedded and mobile platforms
  – Record and replay using hardware-software (Sensys 11)
  – Record and replay using software only (IPSN 15)
Dependability in the Very Small

- Embedded wireless networks have fundamental resource constraints and are often deployed in hostile or uncertain environments
- Constraints include: Energy, Bandwidth, Untrusted nodes, Disconnected networks

Goal:
- Develop middleware that provides a robust platform keeping environment constraints in mind
- Provide detection, diagnosis, and isolation functionality

Solution directions:
- Software for tracing events on the wireless node [Startup company: SensorHound]
- Replaying traced events on a lab server
- Debugging support from replayed trace
- Fastest reprogramming protocol to upload a patch to the network while nodes are deployed in the field
Embedded System Debugging

- Pre-deployment techniques for debugging embedded wireless systems (EWS)
  - Simulation and emulation (e.g., Cooja, TOSSIM)
  - Formal testing
  - Language enforced safety [SafeTinyOS SenSys07]
- Experience shows not all bugs found pre-deployment
  - 23 vulnerabilities already reported in 2016 to ICS-CERT
  - Clearly need for post-deployment debugging tools
- We focus on post-deployment debugging

Why keep detailed execution information?

- Traditional in-deployment debugging of EWS
  - Ad-hoc printf statements
  - Instrument code to log specific events
- Alternative approach is record and replay
  - Captures remote execution (control flow and memory)
  - Faulty execution can be repeatedly replayed in emulator
Problem statement

- How to perform tracing and profiling of software
  - Non-intrusively
  - With high spatial and temporal granularity
  - Low energy
  - Low cost
  - Easy to integrate and deploy
- Tracing provides a sequence of events useful for debugging
- Profiling provides energy consumption and time per event

Roadmap

- Dependability basics
- Debugging in the large: Large-scale distributed applications
  - Using metric mining (DSN 12, SRDS 13)
  - Scale-dependent bugs (HPDC 11, HPDC 13)
- Debugging in the small: Embedded and mobile platforms
  - Record and replay using hardware-software (Sensys 11)
  - Record and replay using software only (IPSN 15)

Solution approach: AVEKSHA

- AVEKSHA is a hardware/software approach
- Exploit on-chip debug module (OCDM)
  - Comes free on most μCs
  - Exposed through JTAG interface
  - Asynchronous with μC operation (therefore non-interfering)
  - Advanced features: complex triggers for breakpoints and watchpoints, store state on trigger

What We Built: The Telos Debug Board

- Connects to embedded node (TI Telos here) I/O pins and JTAG
- Has a μC for initialization and configuration
- Has an FPGA for high speed polling of JTAG
- Three modes of execution with varying levels of intrusiveness, expressivity, and speed: breakpoint, watchpoint, and PC polling
What AVEKSHA Achieved

• Fine-grained visibility into the embedded processor and peripheral states

• Helped us discover bugs in the embedded OS (TinyOS)

Roadmap

• Dependability basics
• Debugging in the large: Large-scale distributed applications
  – Using metric mining (DSN 12, SRDS 13)
  – Scale-dependent bugs (HPDC 11, HPDC 13)
• Debugging in the small: Embedded and mobile platforms
  – Record and replay using hardware-software (Sensys 11)
  – Record and replay using software only (IPSN 15)

Overarching Premise

*Maintain as much detailed information as possible about the execution on the embedded devices in situ*

Questions:
1. What kind of detailed execution information is needed?
2. How feasible is it to record detailed executions?

Objectives

- Programmer writes normal C code
- TARDIS uses flash storage like a black box recorder
- Debugger faithfully reproduces every instruction and memory access
Challenges

• Must record all sources of non-determinism
  1. Peripheral register reads (e.g., ADC, I2C, and timer)
  2. Interrupt timing

• Resources are constrained

<table>
<thead>
<tr>
<th>Device</th>
<th>Flash</th>
<th>RAM</th>
<th>Clock speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Telos</td>
<td>1 MB</td>
<td>10 kB</td>
<td>8 MHz</td>
</tr>
<tr>
<td>ARM Cortex M4</td>
<td>1 MB</td>
<td>136 kB</td>
<td>204 MHz</td>
</tr>
</tbody>
</table>

• A simple toy app (MHO) generates 12.9 KB/sec of non-determinism, fills flash in about 1 minute
• Solution should not interfere with real-time constraints

Solution approach

• Source-to-source compiler instruments C code to record all non-determinism

• Reduce log size by
  – Carefully identifying only what needs to be recorded
  – Efficient compression techniques tailored to each source of non-determinism

• Contributions
  – First system-level record and replay for sensor networks
  – Demonstrate with both TinyOS and Contiki
Compiler design

- **Peripheral registers**
  - Assumes peripheral registers are addressed statically
  - TARDIS-CIL discovers and instruments all reads in code
  - Creates mapping file so emulator knows how each register is encoded

- **Interrupt timing**
  - Instruction counter not available on \(\mu C\)
  - Software approach records interrupt return address and loop count
  - Loops and functions instrumented to increment loop counter

Runtime design

- Record log into buffers in RAM
- Buffers compressed and written to flash during down time (before sleep is called)
- Flash is a circular buffer, similar to black box recorder
Replay design

- Emulator (mspsim) executes code
- Starts from known state or checkpoint
- Whenever peripheral register read encountered, value is taken from log
- Interrupts delivered at logged time

Roadmap

- Debugging in the large: Large-scale distributed applications
  - Using metric mining (DSN 12, SRDS 13)
  - Scale-dependent bugs (HPDC 11, HPDC 13)
  - Computational genomics (Supercomputing 14, ICS 16)
- Debugging in the small: Embedded and mobile platforms
  - Record and replay using hardware-software (Sensys 11)
  - Record and replay using software only (IPSN 15)
    - Reducing amount of logged information
  - Evaluation
  - Cellular network data analytics (HotDep 15, Movid 16)
Applications

- MHO (Multihop Oscilloscope) – one node and base station
- MHO network – 9 node network
- EM (Earthquake Monitor) – 100Hz samples
- Collect – Contiki OS

Rate of log growth

- “Uncompressed” is recording only non-deterministic registers
- Logging increases with LPL and with radio traffic due to more interrupts and timer events
- Why does TARDIS increase significantly from Network 64ms to Network 512ms?
Average power consumption

- Change in power consumption only noticeable when using LPL
- TARDIS power consumption increases when going from 512ms to 64ms
- Radio is most significant consumer of energy

Case Study: CTP Bug

- When partition forms for longer than 30 seconds, network takes 30 minutes to heal
- Goes against CTP principal of quick recovery from broken links
- ETX value continues to rise because of routing loops in partitioned nodes
- Because TARDIS can replay all memory state we can see ETX increasing
- Logging rate is low in healthy network but raises due to high traffic caused by routing loops
TARDIS Demo

- Demo presented to IPSN 2015
- Simple light tracking application
- Demonstrates GDB like features

Take-Aways: Debugging Embedded Systems

- It is possible to perform record and replay for embedded devices and networks of devices
  - Caveat: Surge of events may overwhelm ability of any solution to record faithfully
  - Hardware-software approach relies on fast FPGA
  - Software approach relies on careful understanding of the semantics of the data and state registers
- Record and replay is useful for debugging
  - AVEKSHA useful for debugging fine-grained bugs
  - Both can be used in gdb like mode since exact memory state is recreated during replay
Evolving Issues in Dependability and Data Analytics

• Dependability in the large
  – Take scale and data-dependence into account
  – My interests: Large-scale clusters and distributed applications; Computational genomics

• Dependability in the small
  – Close interactions with physical environment will expose new failure modes
  – Constrained resources of bandwidth and energy will require adaptation of existing dependability solutions
  – My interests: Embedded wireless networks; Smart grid

• Commonalities: Data analytics for dependable systems
  – Need access to (almost) real time data
  – Predictive analytics is crucially important
  – Lightweight analytics, both in training and in production mode
  – Streaming analytics geared toward data in motion

Credits:
Mostafa Ammar (Georgia Tech); Fahad Arshad (Purdue); Somali Chaterji (Purdue); Matthew Creti (SensorHound); Patrick Eugster (Darmstadt); Sajjad Hossain (Purdue); Kaustubh Joshi (AT&T); Michael Kistler (IBM); Milind Kulkarni (Purdue); Ignacio Laguna (LLNL); Kanak Mahadik (Purdue); Tarun Mangla (Georgia Tech); Samuel Midkiff (Purdue); Rajesh Panta (AT&T); Vijay Raghunathan (Purdue); Vinai Sundaram (SensorHound); Nawanol Theera (Purdue); Ellen Zegura (Georgia Tech); Bowen Zhou (LinkedIn)

Presentation available from: engineering.purdue.edu/dcsl
Backup Slides

Costly Software Bugs in History

- Facebook IPO: May 2012
  - NASDAQ earmarked $62 million for reimbursing investors and says it expects to incur significant costs beyond that for system upgrades and legal battles. WSJ pegged the total loss to investors at $500M.
  - Investors were unsure how much of Facebook they’d bought. There were 12 million postponed share orders suddenly filled between 1:49 p.m. and 1:51 p.m. without being properly marked ‘late sale’, which exaggerated the impression that people were trying to dump Facebook shares.
  - **Diagnosis:** Computer systems used to establish the opening price were overwhelmed by order cancellations and updates during the "biggest IPO cross in the history of mankind," Nasdaq Chief Executive Officer Robert Greifeld said Sunday. Nasdaq's systems fell into a "loop" that prevented it from opening the shares on schedule.
Costly Software Bugs in History

  – 50 million people lost power for up to two days in the biggest blackout in North American history. The event contributed to at least 11 deaths and cost an estimated $6 billion.
  – **Diagnosis:** The task force responsible for investigating the cause of the blackout concluded that a software failure at FirstEnergy Corp. "may have contributed significantly" to the outage.
  – FirstEnergy’s Alarm and Event Processing Routine (AEPR), a key software program that gives operators visual and audible indications of events occurring on their portion of the grid, began to malfunction. As a result, "key personnel may not have been aware of the need to take preventive measures at critical times”.
  – Internet links to Supervisory Control and Data Acquisition (SCADA) software weren't properly secure and some operators lacked a system to view the status of electric systems outside their immediate control.

Costly Software Bugs in History

• Medical Devices: Ongoing
  – It has been shown to be possible for a heart defibrillator and pacemaker to reprogram it to shut down and to deliver jolts of electricity that could be fatal. (For a device called Maximo from industry #1 company called Medtronic)
  – Also possible to glean personal patient data by eavesdropping on signals from the tiny wireless radio embedded in the implant as a way to let doctors monitor and adjust it without surgery.
  – 1983-1997: There were 2,792 quality problems that resulted in recalls of medical devices, 383 of which were related to computer software (14%), according to a 2001 study analyzing FDA reports of the medical devices that were voluntarily recalled by manufacturers.
  – 2015: FDA has had 86 recalls, categorized as ones where there is “reasonable probability that use of these products will cause serious adverse health consequences or death.” At least 12 of the recalls were likely caused by software defects.
Effect of major network outages on large business customers

October 2000 data

- Survey by Meta Group Inc. of 21 industrial sectors in 2000 found the mean loss of revenue due to computer system downtime was $1.01M/hour
Thrust #1: Distributed Intrusion Tolerant System

- Distributed systems subjected to malicious attacks to services, including hitherto unknown attacks
- Objective is to tolerate intrusions, not just detect
- Different phases: Detection, Diagnosis, Containment, Response.
- Solution approach:
  - Attack graph based modeling of multi-stage attacks
  - Algorithms for containment and survivability computation and semi-optimal response selection
  - Semi-optimal selection and configuration of intrusion detection sensors for gaining visibility into the security state of system
  - Dealing with zero-day attacks

Predictive Reliability Engine for Cellular Networks

Thrust #1: Distributed Intrusion Tolerant System

- FTP Server
- Web Server
- E-Mail Server
- Application Server
- Media Server
- Directory Server
- File Server
- Management Server
- Database Server
- Hacker
• TARDIS performs some compression and writes to flash during downtime, when node would be sleeping
• This increases duty cycle and consequently power consumption
• Unmodified Network 512ms increases over Network 64ms due to longer active time to send and receive messages
• TARDIS Network 64ms increases over Network 512ms due to longer radio on time to perform clear channel assessment

Example of Scale-dependent Bugs

• A bug in MPI_Allgather in MPICH2-1.1
  – Allgather is a collective communication which lets every process gather data from all participating processes
Example of Scale-dependent Bugs

- MPICH2 uses distinct algorithms to do Allgather in different situations
- Optimal algorithm is selected based on the total amount of data received by each process

```c
int MPIR_Allgather (  
    int recvcount,  
    MPI_Datatype recvtype,  
    MPID_Comm *comm_ptr  
)  
{  
    int comm_size, rank;  
    int curr_cnt, dst, type_size, left, right, jnext, comm_size_is_pof2;  
    ...  
    if ((recvcount*comm_size*type_size < MPIR_ALLGATHER_LONG_MSG)  
        && (comm_size_is_pof2 == 1)) {  
        /* Short or medium size message and power-of-two no. of processes.  
         * Use recursive doubling algorithm */  
        ....  
    } else if (recvcount*comm_size*type_size < MPIR_ALLGATHER_SHORT_MSG) {  
        /* Short message and non-power-of-two no. of processes. Use  
         * Bruck algorithm (see description above). */  
        ....  
    } else {  
        /* Long message or medium-size message and non-power-of-two  
         * no. of processes. Use ring algorithm. */  
        ....  
    }  
}  
```
Roadmap

• Debugging in the large: Large-scale distributed applications
  – Using metric mining (DSN 12, SRDS 13)
  – Scale-dependent bugs (HPDC 11, HPDC 13)
  – Computational genomics (Supercomputing 14, ICS 16)
• Debugging in the small: Embedded and mobile platforms
  – Record and replay using hardware-software (Sensys 11)
  – Record and replay using software only (IPSN 15)
  – Reducing amount of logged information
  – Evaluation
  – Cellular network data analytics (HotDep 15, Movid 16)

Domain-specific & Lightweight Compression

1. Non-determinism of registers
2. Polling loops
3. Register masking pattern
4. Sleep-wake cycling and interrupts
5. Timer registers
6. State registers
7. Data registers
1. Only record non-determinism

- Not all peripheral registers are non-deterministic
- In some registers only particular bits are non-deterministic
- Record only the non-deterministic bits, reduces log by 26.8%

2. Polling loops

- Example, interrupt register checked until transmitting flag is cleared
- TARDIS-CIL identifies loops that have no side effect on execution
- We assume polling loops are eventually exited
- Therefore, no need to record peripheral register reads in polling loops, reduces log by 25.9%
6. State registers

pending_interrupts = IFG;

- State registers report a state, for example, interrupt flags indicating pending interrupt
- Consecutive reads often repeat value
- Design: encode state registers with RLE, reduces state log by 47.8%

7. Data registers

receive_byte = RXBUF;

- Example: I2C data
- Comes from radio and sensors
- Design: compression using light-weight generic compression LZRW-T
- Reduces data log by 65.7%