How Does Software Fail and What Should be Done About It?

DCIT; Nov. 2015

Prof. Kishor Trivedi

Duke High Availability Assurance Lab (DHAAL)
Department of Electrical and Computer Engineering
Duke University, Durham, NC 27708-0291
E-mail: ktrivedi@duke.edu URL: www.ee.duke.edu/~ktrivedi
Duke High Availability Assurance Laboratory (DHAAL)

Kishor Trivedi
Dept. of Electrical & Computer Engineering
Duke University
Email: ktrivedi@duke.edu
URL: www.ee.duke.edu/~ktrivedi

Internationally connected with groups in USA, Germany, Italy, Japan, China, Brazil, New Zealand and Spain

Copyright © 2015 by K.S. Trivedi
DHAAL

- Dhaal means “Shield” in Hindi/Gujarati
- DHAAL research is about shielding systems from various threats
Books Update


- Queuing Networks and Markov Chains, 1998

- Reliability and Availability Engineering, Cambridge University Press, 2016 (green book)
Outline

- Motivation
- Real System Examples
- Software Fault Classification
- Environmental Diversity
- Methods of Mitigation
- Software Aging and Rejuvenation
- Conclusions
Pervasive Dependence on Computer Systems
Need for High Reliability/Availability

Communication

Health & Medicine
Entertainment
Avionics
Banking

Copyright © 2015 by K.S. Trivedi
Steady-state availability ($A_{ss}$) or just availability

Long-term probability that the system is available when requested:

$$A_{ss} = \frac{MTTF}{MTTF + MTTR}$$

MTTF is the system mean time to failure, a complex combination of component MTTFs.

MTTR is the system mean time to recovery.

May consist of many phases.
Basic Definitions

- Downtime in minutes per year
  - In industry, (un)availability is usually presented in terms of annual downtime.
  - Downtime = $8760 \times 60 \times (1 - A_{ss})$ minutes.
  - In Industry it is common to define the availability in terms of number of nines
    - 5 NINES ($A_{ss} = 0.99999$) $\Rightarrow$ 5.26 minutes annual downtime
    - 4 NINES ($A_{ss} = 0.9999$) $\Rightarrow$ 52.56 minutes annual downtime
49% of Fortune 500 companies experience at least 1.6 hours of downtime per week

Approx. 80 hours/year = 4800 minutes/year

\[ A_{ss} = \frac{8760 - 80}{8760} = 0.9908 \]

That is, between 2 NINES and 3 NINES!

This study assumes planned and unplanned downtime, together
Some real examples from High Tech companies

Jan. 2014, Gmail was down for 25 – 50 min.

Oct. 2013, Unavailable services like post photos and “likes”

Feb. 2013, Windows Azure down for 12 hours

Jan. 2013, AWS down for an hour approx.

Sept. 2012 - GoDaddy (4 hours and 5 millions of websites affected)

Copyright © 2015 by K.S. Trivedi
More examples of failures

Oct. 2012 - Amazon Webservices - 6 hours (Memory leak)
Amazon EC2 - 2 hours

Sept. 2011 - Google Docs service outage (1 hour) - A memory leak due to a software update

Sept. 2011 - Microsoft Cloud service outage (2.5 hours)

These examples indicate that even the most advanced tech companies are offering less than five NINES of availability

• And only considering one failure!!!!
Software is the problem

Jim Gray’s paper titled “Why do computers stop and what can be done about it?” pointed out this trend in 1985, followed by his paper “A census of tandem system availability between 1985 and 1990”
High Reliability/Availability: Software is the problem

- Hardware fault tolerance, fault management, reliability/availability modeling relatively well developed
- System outages more due to software faults

Key Challenge:

Software reliability is one of the weakest links in system reliability/availability
Increasing SW Failure Rate?
Planetary Missions Flight Software: A. Nikora of JPL

The interval between the first and last launch: 8.76 years.

The interval between successive launches ranges from: 23 to 790 days.
Today’s complex systems (including CPS and IoT) are mostly software with some hardware thrown in. **Software failures are a major cause of system undependability.**

The focus so far has been on software faults; we need to pay attention to failures caused by software and the recovery from these failures. Or focus so far has been on software reliability; we need to pay attention to software availability as well.

**Software failures during operation are a fact that we need to learn to deal with.** Traditional method of software fault tolerance based on design diversity is expensive and hence does not get used extensively. Software fault tolerance based on **inexpensive environmental diversity should be exploited.**
Software Reliability: Known Means

- Fault prevention or Fault avoidance
- Fault Removal
- Fault Tolerance
Software Reliability

- **Fault prevention or Fault avoidance**
  - Good software engineering practices
    - Requirement Elicitation (Abuse Case Analysis – TCS SSA)
    - Design Analysis / Review
    - Secure Programming Standard & Review
    - Secure Programming Compilation
    - Software Development lifecycle
    - Automated Code Generation Tools (IDE like Eclipse)
  - Use of formal methods
    - UML, SysML, BPM
    - Proof of correctness
    - Model Checking (SMART, SPIN, PRISM)

- Bug free code not yet possible for large scale software systems
  - Impossible to fully **test** and verify if software is fault-free
    - “Testing shows the presence, not the absence, of bugs”
      - E. W. Dijkstra

- Yet there is a strong need for failure-free system operation
Software Reliability

- Fault prevention or Fault avoidance
- Fault Removal
- Fault Tolerance
Fault removal

- Can be carried out during
  - the specification and design phase
  - the development phase
  - the operational phase

- Failure data may be collected and used to parameterize a software reliability growth model (SRGM) to predict when to stop testing.

Software is still delivered with many bugs either because of inadequate budget for testing, very difficult to reproduce/detect/localize/correct bugs or inadequacy of techniques employed/known.
Software Reliability

- Fault prevention or Fault avoidance
- Fault Removal
- Fault Tolerance
High Reliability/Availability: Software is the problem

**Software fault tolerance** is a potential solution to improve software reliability in lieu of virtually impossible fault-free software.
Software Fault Tolerance

Classical Techniques

Design diversity

- N-version programming
- Recovery block
Software Fault Tolerance
Classical Techniques

- N-version programming
- Recovery blocks
- ...

Design diversity

Expensive ⇒ not used much in practice!

Yet there are stringent requirements for failure-free operation

Challenge: Affordable Software Fault Tolerance
Today's complex systems (including CPS and IoT) are mostly software with some hardware thrown in. **Software failures are a major cause of system undependability.**

The focus so far has been on software faults; **we need to pay attention to failures caused by software and the recovery from these failures.** Or focus so far has been on software reliability; we need to pay attention to software availability as well.

**Software failures during operation are a fact that we need to learn to deal with.** Traditional method of software fault tolerance based on design diversity is expensive and hence does not get used extensively. Software fault tolerance based on **inexpensive environmental diversity should be exploited.**
Outline

- Motivation
- Real System Examples
- Software Fault Classification
- Environmental Diversity
- Methods of Mitigation
- Software Aging and Rejuvenation
- Conclusions
Real Systems
Example of a real systems and their High availability implementations
High availability SIP Application Server Configuration on IBM WebSphere

More details in PRDC 2008 and ISSRE 2010 papers

Copyright © 2015 by K.S. Trivedi
High availability SIP Application Server configuration on WebSphere

- **Hardware configuration:**
  - Two BladeCenter chassis; 4 blades (nodes) on each chassis (1 chassis sufficient for performance)

- **Software configuration:**
  - 2 copies of SIP/Proxy servers (1 sufficient for performance)
  - 12 copies of WAS (6 sufficient for performance)
  - Each WAS instance forms a redundancy pair (replication domain) with WAS installed on another node on a different chassis

The system has hardware redundancy and software redundancy.
High availability SIP Application Server configuration on WebSphere

- **Software Fault Tolerance**
  - Identical copies of SIP proxy used as backups *(hot spares)*
  - Identical copies of WebSphere Applications Server (WAS) used as backups *(hot spares)*
  - **Type of software redundancy** – (not design diversity) but replication of identical software copies
  - **Normal recovery after a software failure**
    - Restart software, reboot node or fail-over to a software replica; only when all else fails, a “software repair” is invoked
Software Fault Tolerance: New Thinking

1. Have been known to help in dealing with hardware transients.
2. Do they help in dealing with failures caused by software bugs?
3. If yes, why?
Software Fault Tolerance: New Thinking

Failover to an identical software replica (that is not a diverse version)

Twenty years ago this would be considered crazy!

Does it help?

If yes, why?
Outline

- Motivation
- Real System Examples
- **Software Fault Classification**
- Environmental Diversity
- Methods of Mitigation
- Software Aging and Rejuvenation
- Conclusions

Copyright © 2015 by K.S. Trivedi
Software Faults
main threats to high reliability, availability & safety
Failure occurs when the delivered service no longer complies with the desired output.

Error is that part of the system state which is liable to lead to subsequent failure.

Fault (or bug) is adjudged or hypothesized cause of an error.

Faults are the cause of errors that may lead to failures.
Need to Classify bug types

- We submit that a software fault tolerance approach based on *retry*, *restart*, *reboot* or *fail-over* to an identical software replica (not a diverse version) works because of a significant number of software failures are caused by **Mandelbugs** as opposed to the traditional software bugs now called **Bohrbugs**.
Need to Classify bug types

- For over two decades, researchers have reported the phenomenon of “software aging”.
  - i.e., degraded performance and/or increased failure rate of long-running software systems.

- Puzzle: How can performance and failure rate change if the software code is not modified?!

⇒ Study software fault types and their relationships
**A Classification of Software Faults**

**Bohrbug :=** A fault that is easily isolated and that manifests consistently under a well-defined set of conditions, because its activation and error propagation lack complexity.

Example: A bug causing a failure whenever the user enters a negative date of birth

- Since they are easily found, Bohrbugs may be detected and fixed during the software testing phase.
- The term alludes to the physicist Niels Bohr and his rather simple atomic model.
Mandelbug – Definition

- **Mandelbug** := A fault whose activation and/or error propagation are complex. Typically, a Mandelbug is difficult to isolate, and/or the failures caused by it are not systematically reproducible.

Example: A bug whose activation is scheduling-dependent:
- The residual faults in a thoroughly-tested piece of software are mainly Mandelbugs.
- The term alludes to the mathematician Benoît Mandelbrot and his research in fractal geometry.
- Sometimes called concurrency or non-deterministic or soft bugs; failures resulting from these bugs are called transient failures.
A fault is a Mandelbug if its manifestation is subject to the following complexity factors:

- Long time lag between fault activation and failure appearance
- Operating environment (OS, other applications running concurrently, hardware, network…)
- Timing among submitted operations
- Sequencing or ordering of operations

A failure due to a Mandelbug may not show up upon the resubmission of a workload if the operating environment has changed enough.
Examples of Types of Bugs in IT Systems


- The selected TCS projects ranged across a number of business systems in the banking, financial, government, IT, pharmacy, and telecom sector.
Mandelbug Reproducibility

- **Mandelbugs** are really hard to reproduce
  - Conducted a set of experiments to study the environmental factors (i.e., disk usage, memory occupation and concurrency) that affect the reproducibility of Mandelbugs
    - High usage of environmental factors increases significantly the reproducibility of Mandelbugs

Aging-related Bug – Definition

- **Aging-related bug** := A fault that leads to the accumulation of errors either inside the running application or in its system-context environment, resulting in an increased failure rate and/or degraded performance.

Example:
- A bug causing memory leaks in the application
- Note that the aging phenomenon requires a delay between (first) fault activation and failure occurrence.
- Note also that the software *appears to age* due to such a bug; there is no physical deterioration.
Bohrbug and Mandelbug are complementary antonyms. Aging-related bugs are a subtype of Mandelbugs.
Important Questions about these Bugs

- What fraction of bugs are Bohrbugs, Mandelbugs and aging-related bugs
  - How do these fractions vary
    - over time
    - over projects, languages, application types,…

- Need of Real Data
Software Faults in IT Systems

  - 61.4% Bohrbugs (BOH)
  - 32.1% non-aging-related Mandelbugs (NAM)
  - 4.4% aging-related bugs (ARB)
  - 2.1% faults of unknown type (UNK)


<table>
<thead>
<tr>
<th>Project</th>
<th>LoC (Considered)</th>
<th>% BOH</th>
<th>% NAM</th>
<th>% ARB</th>
<th>% UNK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>1.31M</td>
<td>42.2</td>
<td>41.9</td>
<td>8.3</td>
<td>7.6</td>
</tr>
<tr>
<td>MySQL</td>
<td>453K</td>
<td>56.6</td>
<td>30.3</td>
<td>7.7</td>
<td>5.4</td>
</tr>
<tr>
<td>HTTPD</td>
<td>145K</td>
<td>81.1</td>
<td>10.5</td>
<td>7.0</td>
<td>1.4</td>
</tr>
<tr>
<td>AXIS</td>
<td>80K</td>
<td>92.5</td>
<td>3.5</td>
<td>4.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>
Software Faults and Mitigation Types

- The fault classification is not only theoretical, it has also practical implications.

- Each type of software fault may require different type of approach during development, testing, as well as during operations.
Software Faults and Mitigation Types

Failure mitigation techniques

Environmental Diversity

Fault/Error mitigation techniques

Bohrbugs

Mandelbugs

Non-aging-related Mandelbugs

Aging-related bugs

Software faults

Fix/Patch

Workaround

Use as is

Retry

Restart

Reboot

Failover to identical

Rejuvenate

Reconfigure

Failover to nonidentical

Software fault Classification

Copyright © 2015 by K.S. Trivedi
Software Fault and Failure Mitigation Types

- Software fault mitigation types:
  - Fix/Patch

- Software failure mitigation types:
  - Reconfigure
  - Failover to non-identical
  - Retry
  - Restart
  - Reboot
  - Failover to identical

Environmental diversity
Outline

- Motivation
- Real System Examples
- Software Fault Classification
- **Environmental Diversity**
- Methods of Mitigation
- Software Aging and Rejuvenation
- Summary
Environmental diversity
A new thinking to deal with software faults and failures
Software Fault Tolerance: New Thinking

- Environmental Diversity as opposed to Design Diversity

- Our claim is that this (retry, restart, reboot, failover to identical software copy) works since failures due to Mandelbugs are not negligible. We thus have an affordable software fault tolerance technique that we call Environmental Diversity
What is Environmental diversity?

- The underlying idea of Environmental diversity
  - Retry a previously faulty operation and it most likely works -- Why?
  - because of the environment where the operation was executed has changed enough to avoid the fault activation.

- The environment is understood as
  - OS resources, other applications running concurrently and sharing the same resources, interleaving of operations, concurrency, or synchronization.

- This is Fault Tolerance since we do not necessarily fix the fault; fault caused a failure but this failure is dealt with by using time redundancy hence the user does not experience the failure again on retry.
Outline

- Motivation
- Real System Examples
- Software Fault Classification
- Environmental Diversity
- **Methods of Mitigation**
- Software Aging and Rejuvenation
- Summary
OS Availability Model (IBM BladeCenter)

Fix (Failed due to a Bohrbug)

Reboot (Failure due to a Mandelbug)
Outline

- Motivation
- Real System Examples
- Software Fault Classification
- Environmental Diversity
- Methods of Mitigation
- Software Aging and Rejuvenation
- Summary
Software Aging and Rejuvenation

Aging-related Bugs:
Replicate, Restart, Reboot, Rejuvenate
Software Aging

- Aging phenomenon
  - Error conditions accumulating over time

Performance degradation, system failure

- Main causes of Software Aging
  - Memory leak, fragmentation, Unterminated threads, Data corruption, Round-off errors, Unreleased file-locks, etc…

- Observed system
  - OS, Middle-ware, Netscape, Internet Explorer etc
Software Aging - Definition

- "Software Aging" phenomenon
  - Long-running software tends to show an increasing failure rate.
  - \textit{Not} related to application program becoming obsolete due to changing requirements/maintenance.
  - Software appears to age; no real deterioration
Software Aging Examples

Oct. 2012 - Amazon Web Services Outage Caused By Memory Leak And Failure In Monitoring Alarm

Sept. 2011 - Google Docs Outage Blamed on Memory Glitch

Feb. 1991 - The Patriot Missile Software Failure

International Space Station (ISS) FC SSC memory leaks problems
Software Aging – More Examples

- Cisco Catalyst Switch [Matias Jr.]
- File system aging [Smith & Seltzer]
- Gradual service degradation in the AT&T transaction processing system [Avritzer et al.]
- Error accumulation in Patriot missile system’s software [Marshall]
- Resources exhaustion in Apache [Li et al., Grottke et al.]
- Physical memory degradation in a SOAP-based Server [Silva et al.]
- Software aging in Linux [Cotroneo et al.]
- Crash/hang failures in general purpose applications after a long runtime
A Methodology for Detection and Estimation of Software Aging,
S. Garg, A. van Moorsel, K. Vaidyanathan and K. Trivedi.
Software Fault Types & Their Mitigation

- **Software faults**
  - **Bohrbugs**
  - **Mandelbugs**
    - Non-aging-related Mandelbugs
    - Aging-related bugs

- **Fault/Error mitigation techniques**
  - Fix/Patch
  - Workaround
  - Use as is

- **Failure mitigation techniques**
  - Reconfigure
  - Failover to nonidentical

- **Environmental Diversity**
  - Retry
  - Restart
  - Reboot
  - Failover to identical
  - Rejuvenate

Copyright © 2015 by K.S. Trivedi
Software Rejuvenation

- **Software rejuvenation** is a cost effective solution for improving software reliability by avoiding/postponing unanticipated software failures/crashes.

- It allows proactive recovery to be carried either automatically or at the discretion of the user/administrator

Rejuvenation of the environment, not of software
Software Rejuvenation Examples

- **Patriot missile system software** - switch off and on every 8 hours
- **ISS FS SSC (ISS File system)** - switch off and on every 2 months

Tens of US Patents related with this technology

Process and connections restart/recycling

Copyright © 2015 by K.S. Trivedi
Software Rejuvenation: More Examples

For more real examples:

Software Rejuvenation – Trade-off

- Advantages
  - Reduces costs of sudden aging-related failures
  - Can be applied at the discretion of the user/administrator

- Disadvantages
  - Direct costs of carrying out rejuvenation
  - Opportunity costs of rejuvenation (downtime, decreased performance, lost transactions etc)

**Important research issue:**
Find optimal times to perform rejuvenation!
Software Rejuvenation - Approaches

Two approaches based on WHEN:

- Time-Based rejuvenation approaches
  - Rejuvenation applied regularly and at predetermined time intervals.
  - Widely used in real environments
    - Web servers (Apache)
    - ISS two-months reboot
    - Telecommunication systems
Software Rejuvenation - Approaches

- Two approaches based on **WHEN**:  
  - Measurement (or inspection)-based rejuvenation  
    - System metrics continually monitored  
    - Threshold based: Rejuvenation triggered when the crash is imminent based on the observation  
    - Predictive: measurements are used to predict the time to resource exhaustion or time to failure
Software Rejuvenation Scheduling

Rejuvenation strategies

- Time-based
- Inspection-based
  - Threshold-based
  - Prediction-based
Software Rejuvenation – Approaches

- Two approaches based on HOW:
  - Use analytical model to optimize rejuvenation schedule
  
  - Lucent Bell Labs [Huang et al., ‘95]
  - Duke [IEEE-TC’98, SIGMETRICS’96, ISSRE’95, PRDC’00, SIGMETRICS’01, Comp J.’01, SRDS’02, DSN’02, ISSRE’02, DSN’03, IEEE-TR’05]
  - Others [IPDS’98, PNPM’99]
Time-Based Rejuvenation

- **Available**
  - **Gen\(g_f\)**
  - **F(t)**
  - **Det(\(\delta\))**
  - **Gen\(g_r\)**

Graph:
- **B**: Failure
- **A**: Available
- **C**: Rejuvenation

Distribution of TTF Needed to determine optimal value of rejuvenation trigger

- **state A**: the system is up and available
- **state C**: the system is under software rejuvenation
- **state B**: the system is down and under reactive repair
Purpose of Measurements

- **Times to Failure:** to parameterize the analytic model discussed earlier
  - **Fit the measured data to a known distribution and then use the model to find optimal rejuvenation schedule**
  - **Can Speed up measurements** by using ALT (accelerated life tests) and ADT (accelerated degradation tests) [Matias et al; IEEE-TR 2010; ISSRE 2010], if measurements are done in a controlled environment, or by using simulation with IS (importance sampling) [Zhao et al; PE2013, ISSRE 2011, JETC 2014]
  - Taking the measured sequence directly to determine optimal rejuvenation schedule **without fitting** measured times to failure to a distribution [Dohi et al. PRDC 2000]
Purpose of Measurements

- Measuring performance variables
  - to predict time to resource exhaustion or time to failure
  - to trigger rejuvenation in a threshold based scheme [discussed here in detail]
Software Rejuvenation Granularities

- **Physical node rejuvenation granularity**
  - Fast OS rejuvenation
  - OS component rejuvenation

- **Operating system rejuvenation granularity**
  - VM checkpointing
  - VM live-migration

- **Virtual machine monitor rejuvenation granularity**
  - Application checkpoint restart
  - Application replication/redundancy

- **Virtual machine rejuvenation granularity**

- **Application rejuvenation granularity**

- **Application component rejuvenation granularity**

**Bypass-based approaches**

**Reduced-based approaches**
IBM xSeries
Software Rejuvenation Agent (SRA)

- IBM Director system management tool
  - Provides GUI to configure SRA
  - Acts upon alerts

- Two versions
  - Periodic rejuvenation
  - Prediction-based rejuvenation
Outline

- Motivation
- Real System Examples
- Software Fault Classification
- Environmental Diversity
- Methods of Mitigation
- Software Aging and Rejuvenation
- Summary
Summarizing the talk
Summary

- It is possible to enhance software availability during operation exploiting environmental diversity.
- Multiple types of recovery after a software failure can be judiciously employed: restart, failover to a replica, reboot and if all else fails repair (patch).
Summary

- Software aging not anecdotal – real life scientific phenomenon

- Rejuvenation implemented in several special purpose applications and many general purpose cluster systems
Today's complex systems (including CPS and IoT) are mostly software with some hardware thrown in. Software failures are a major cause of system undependability.

The focus so far has been on software faults; we need to pay attention to failures caused by software and the recovery from these failures. Or focus so far has been on software reliability; we need to pay attention to software availability as well.

Software failures during operation are a fact that we need to learn to deal with. Traditional method of software fault tolerance based on design diversity is expensive and hence does not get used extensively. Software fault tolerance based on inexpensive environmental diversity should be exploited.
Thank you for your attention

Kishor Trivedi
Dept. of Electrical & Computer Engineering
Duke High Availability Assurance Lab (DHAAL)
Duke University
ktrivedi@duke.edu
www.ee.duke.edu/~ktrivedi
Key References

Motivation

Key References

Real System


Fault Classification

Key References

Environmental Diversity and Methods of Mitigation

Key References

Software Aging and Software Rejuvenation (1)

Key References

Software Aging and Software Rejuvenation (2)

Key References

**Software Aging and Software Rejuvenation (3)**