Software Fault Tolerance via Environmental Diversity

Kishor Trivedi
Dept. of Electrical & Computer Engineering
Duke High Availability Assurance Lab (DHAAL)
Duke University
Durham, NC 27708
ktrivedi@duke.edu
www.ee.duke.edu/~ktrivedi

SERE
July 2, 2014

Copyright © 2014 by K.S. Trivedi
Outline

- Motivation
- A Real System
- Software Fault Classification
- Environmental Diversity
- Methods of Mitigation
- Software Aging and Rejuvenation
- Conclusions
Pervasive Dependence on Computer Systems $\rightarrow$ Need for High Reliability/Availability
Basic Definitions

- **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.

  - 5 NINES ($A_{ss} = 0.99999$) $\rightarrow$ 5.26 minutes annual downtime.
Number of Nines—Reality Check

- 49% of Fortune 500 companies experience at least 1.6 hours of downtime per week
  
  - Approx. 80 hours/year = 4800 minutes/year
  
  - $A_{ss} = (8760 - 80)/8760 = 0.9908$
  
  - That is, between 2 NINES and 3 NINES!
  
- This study assumes planned and unplanned downtime, together
Achieving High Availability is a Challenge

- 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.
More Failures

- **Black Sept. 2011, In the same week!!!!:**
  - Microsoft Cloud service outage (2.5 hours)
  - Google Docs service outage (1 hour)
    - A memory leak due to a software update

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

- **Oct. 2012 Amazon**
  - 10/15/2012 Webservices – 6 hours (Memory leak)
  - 10/27/2012 EC2 – > 2 hours
Downtown Costs per Hour

- Brokerage operations $6,450,000
- Credit card authorization $2,600,000
- eBay (1 outage 22 hours) $225,000
- Amazon.com $180,000
- Package shipping services $150,000
- Home shopping channel $113,000
- Catalog sales center $90,000
- Airline reservation center $89,000
- Cellular service activation $41,000
- On-line network fees $25,000
- ATM service fees $14,000

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
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.”
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.
Increasing SW Failure Rate?
Planetary Missions Ground Support 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.
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)

- Bug free code not yet possible for large scale software systems

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

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

**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 to predict when to stop testing

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

- Fault prevention or Fault avoidance
- Fault Removal
- Fault Tolerance
There are stringent requirements for failure-free operation of software-based systems – next idea

**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
- N-self check programming

Expensive (unless based on component-based design) → not used much in practice!

Yet there are stringent requirements for failure-free operation

Challenge: Affordable Software Fault Tolerance
Outline

- Motivation
- A Real System
- Software Fault Classification
- Environmental Diversity
- Methods of Mitigation
- Software Aging and Rejuvenation
- Conclusions
High availability SIP Application Server Configuration on IBM WebSphere

PRDC 2008 and ISSRE 2010 papers
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
Escalated levels of Recovery (Telco)

The flowchart briefly depicts the actions taken for recovery after a failure is detected. Try the simplest recovery method first, then a more complex etc.
Software Fault Tolerance: New Thinking

Retry, restart, reboot!

- Known to help in dealing with hardware transients

- Do they help in dealing with failures caused by software bugs?

- If yes, why?
Software Fault Tolerance: New Thinking

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

- Does it help?
- If yes, why?

Twenty years ago this would be considered crazy!
Outline

- Motivation
- A Real System
- Software Fault Classification
- Environmental Diversity
- Methods of Mitigation
- Software Aging and Rejuvenation
- Conclusions
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** 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**
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 bugs.
Mandelbugs complexity factors

- 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
  - Submitted to ISSRE 2014
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 fault activation and failure occurrence.
  - Note also that the software appears to age due to such a bug; there is no physical deterioration
Relationships

- 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 Measurements
  - NASA/JPL Project with Allen Nikora & Michael Grottke & Javier Alonso; results:
    - 52% Bohrbugs
    - 35% Mandelbugs (non-aging-related)
    - 4% Aging-related bugs
    - 7% Operator related
    - 2% Unclassified

- Very similar results for Linux, MySQL, Apache AXIS, httpd
Outline

- Motivation
- A Real System
- Software Fault Classification
- Environmental Diversity
- Methods of Mitigation
- Software Aging and Rejuvenation
- Conclusions
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.
Outline

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

Software (OS, middleware, applications)

Bohrbugs

- Debug/Test
- Design diversity
- Data diversity
Bohrbugs: Remove

- Find and fix the bugs during testing
- Failure data collected during testing
- Calibrate a *software reliability growth model* (SRGM) using failure data; this model is then used for prediction
- Many SRGMs exist (JM, NHPP, HGRGM, etc.)
  - Books by Lyu, Musa, Cai
- Measurements → Empirical (*statistical*) models
Mitigation

Software (OS, middleware, applications)

Bohrbugs

Mandelbugs

Debug/Test
Design diversity
Data diversity
Retry operation
Failover to standby
Restart Application
Reboot Node
OS Availability Model (IBM BladeCenter)

Reboot (Failure due to a Mandelbug)

Fix (Failed due to a Bohrbug)
Markov Availability Model for a Single Replication Domain

- **Failure detection**
  - By WLM
  - By Node Agent
  - Manual detection

- **Recovery**
  - WLM
    - Failover
  - Node Agent
    - Auto process restart
  - Manual recovery
    - Process restart
    - Node reboot
    - Repair
Outline

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

*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
Measurements Showing Resource Exhaustion or Depletion

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 (OS, middleware, applications)

- Bohrbugs
- Mandelbugs
  - Non-Aging-Related Bugs
  - Aging-Related Bugs

Debug/Test
- Design diversity
- Data diversity

Environmental Diversity
- Retry operation
- Failover to standby
- Restart Application
- Reboot Node
- Rejuvenation
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

- Process and connections restart/recycling

- Tens of US Patents related with this technology
Software Rejuvenation
More Examples

- AT&T billing applications [Huang et al.]
- On-board preventive maintenance for long-life deep space missions (NASA’s X2000 Advanced Flight Systems Program) [Tai et al.]
- IBM Director Software Rejuvenation (x-series) [IBM & Duke Researchers]

For more 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

Rejuvenation strategies

- Time-based
- Inspection-based
  - Threshold-based
  - Prediction-based
Software Rejuvenation Granularities

- Physical node rejuvenation granularity
  - Operating system rejuvenation granularity
    - Fast OS rejuvenation
    - OS component rejuvenation
      - Bypass-based approaches
      - Reduced-based approaches
    - 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

Copyright © 2014 by K.S. Trivedi
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
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
Key References


Key References


Key References
