Is Persistent Memory Persistent?

(or, How to Test Against Power Failures)

Terence Kelly and Haris Volos

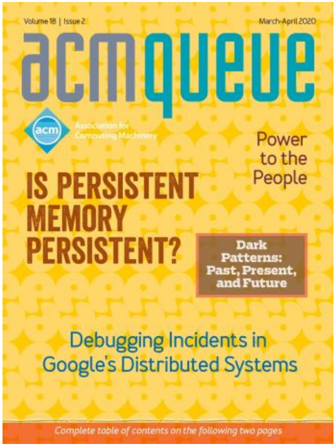
tpkelly@eecs.umich.edu
hvolos01@cs.ucy.ac.cy

SNIA Storage Developer Conference
recorded 9 September 2020

Topic

- Stress-testing hardware & software against power failures
  - persistent memory
  - relational database management systems
  - anything that must survive power failures
- How to test
  - cost-effectively
  - thoroughly
  - convincingly

Further Reading


Volume 18 | Issue 2 March-April 2020

acmqueue

acm Association for Computing Machinery

Power to the People

IS PERSISTENT MEMORY PERSISTENT?

Dark Patterns: Past, Present, and Future

Debugging Incidents in Google's Distributed Systems

Complete table of contents on the following two pages

- *Communications of the ACM*, September 2020
  - hard copy delivered to ACM members (if USPS still works)
  - ACM Digital Library (paywall) `dl.acm.org`
- ACM *Queue* magazine, March/April 2020
  - no paywall `queue.acm.org`

Data Integrity

- Protecting application data is Job One
- Threat: crashes
  - application process crash
  - OS kernel panic
  - power failure
- Power failure *during update*
  - corrupt data
  - destroy data by corrupting metadata
  - e.g., bank transfer creates/destroys money

Failure-Atomic Update

- Applications update data atomically w.r.t. failure
- Evolve data from one consistent state to the next
  - post-crash, restore data to one or the other
  - application recovers from consistent state
- Examples
  - ACID transactions in RDBMS
  - transactional key-value stores
  - new mechanisms for persistent memory

Failures of Failure-Atomicity & Persistence

- Zheng et al., "Torturing Databases," OSDI '14
  - ACID transactions in RDBMSes aren't ACID
  - transactional key-value stores aren't transactional
  - they're all broken
- Zheng et al., "SSDs Under Power Fault," FAST '13
  - SSDs lose data
- Casts doubt on *all* hardware & software
  - especially new stuff

Performance Optimization: Threat or Menace?

- Fast parachutes
  - Spectre/Meltdown: fast CPUs
  - Rowhammer: fast, dense DRAM
- Performance benchmarks abound
- *Integrity benchmarks?*
- Industry makes bad choices that hurt us

What To Do?

- Can’t trust software or hardware beneath application
- But you can test it
- Train as you would fight
  - test against realistic failures
- Sudden whole-system power interruptions
- Big picture: reliability & assurance

Less Strenuous Tests

- `$ kill -9 [pid]` and `raise(SIGABRT);`
  - OS & hardware unaffected
- `$ shutdown now` and injected OS kernel panic
  - hasty but orderly shutdown
  - hardware unaffected
- Management layer power-off
  - gentler than genuine power disconnection
- Gold standard: sudden whole-system power interruptions

Outline

- Motivation
- Power-fail testbed
- Persistent memory tests
- RDBMS tests (breaking news / exclusive!)

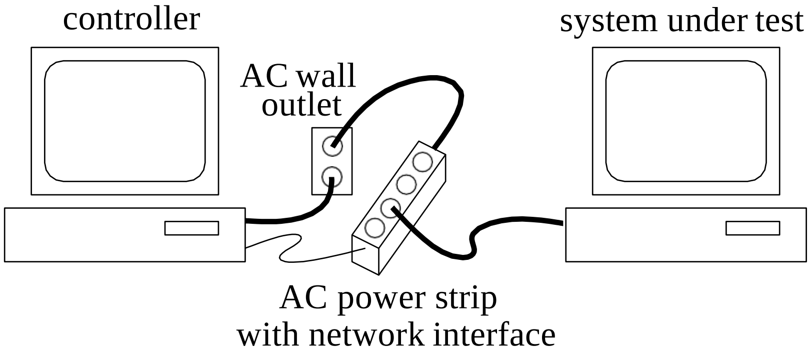
Powerfail Testbed Requirements

- Machine hosting h/w & s/w under test *cuts its own power*
  - allows precise injection of power outages
  - e.g., between every pair of semicolons in critical code
- Host reboots quickly and automatically
  - to start next test cycle
- Host is rugged
  - survives many thousands of off/on cycles
- Testbed is cheap
  - every starving student needs one
  - successful results are more compelling

Host Computer: Bad Choices

- Rented server (“cloud”)
  - unreal: mummified in virtualization
  - customer software can’t cut power to bare metal
- High-end servers
  - management interfaces “power off” too gently
  - expensive
  - real power outages might damage the pricey box
- Laptops
  - lack BIOS features to reboot on restoration of AC power
- Workstations & PCs
  - boot slowly
  - bulky, power-hungry

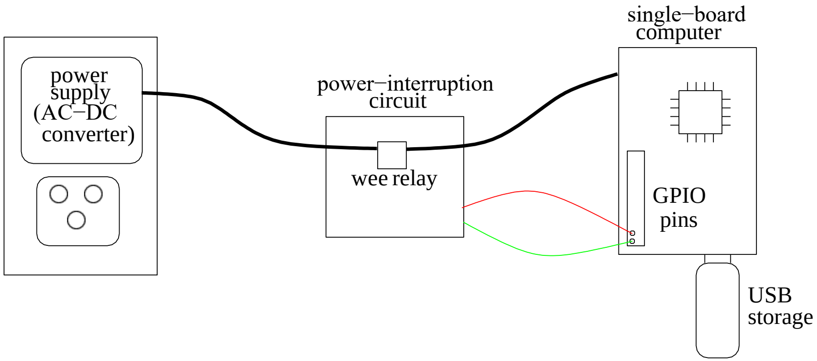
Our Old Testbeds



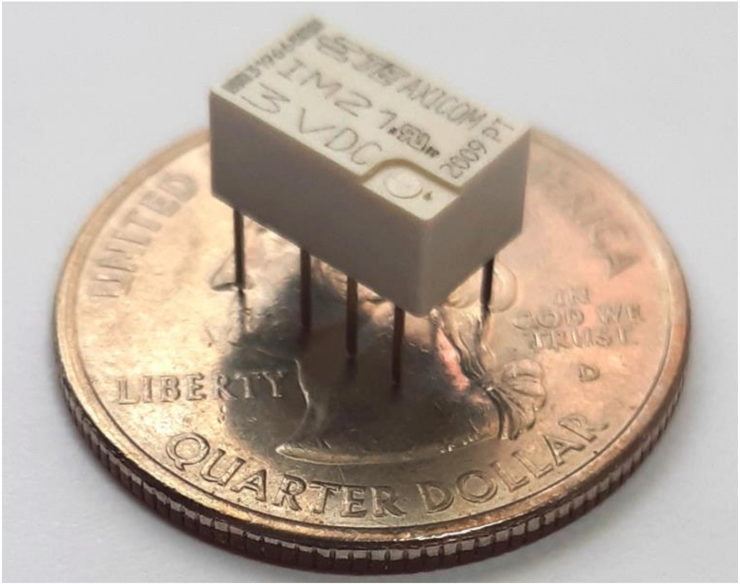
Better Host: Single-Board Computer

- E.g., Raspberry Pi
  - we use model 3B+
- Cheap, expendable
- Rugged
- Runs Linux OS & applications
- Boots quickly and automatically when power restored
- GPIO pins designed to control external circuitry
- Flimsy
  - tests pass on Pi $\Longrightarrow$ likely pass on fancier host

New Testbed


power supply (AC–DC converter)
power–interruption circuit
wee relay
single–board computer
GPIO pins
USB storage

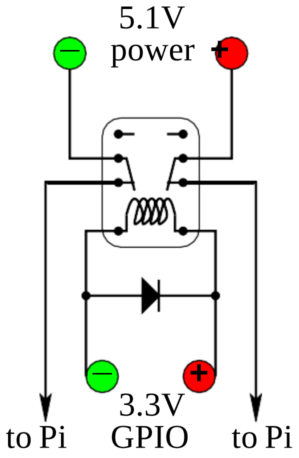
Wee Relay


AXICOM
IM21
3 VDC
2009 PT
LIBERTY
QUARTER DOLLAR
IN GOD WE TRUST
UNITED

Goldilocks and the Three Power Interruption Circuits

- First was too complex
  - *two* relays; dedicated second power supply; IC timer chip; several resistors, capacitors, diodes
- Second was too simple
  - power-off delay too short, not controllable
- Third was just right
  - adjustable power-off delay
  - low component count

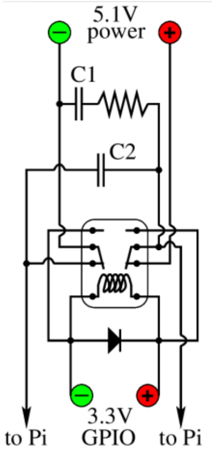
Relay-Only Circuit (Too Simple)



1. Pi's power thru N.C. contacts
2. Software on Pi sets GPIO pin "HI"
3. Relay coil energized
4. Poles jump away from N.C. contacts
5. Power to Pi cut
6. GPIO pin goes "LO"
7. Poles fall back to N.C. position
8. Power to Pi restored
9. Pi reboots

Momentary outage unrealistic?

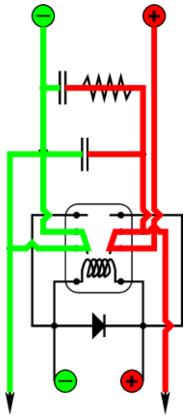
“PiNap” Circuit



(a)

- Use same relay & diode
- Add two capacitors and a resistor
- Route wires thoughtfully

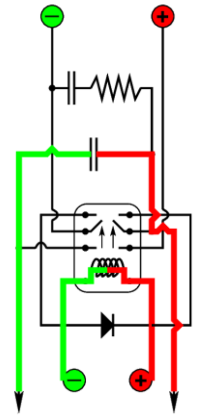
“PiNap” Circuit: Normal Operation


(b)

Pi’s power

- runs thru normally closed contacts
- charges capacitors

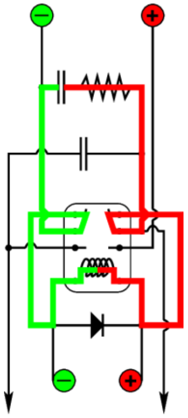
"PiNap" Circuit: Transient



(c)

1. Software on Pi sets GPIO pin "HI"
2. Relay poles in flight N.C. to N.O.
3. Capacitor C2 briefly powers Pi

"PiNap" Circuit: Nap Time

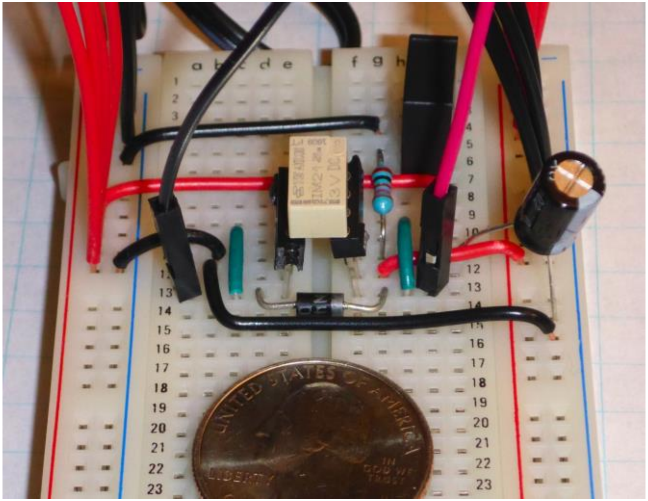


(d)

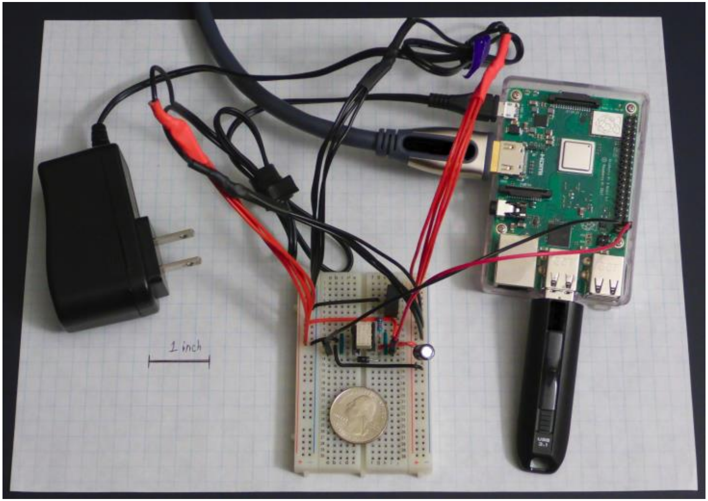
1. Relay poles reach N.O. contacts
2. Capacitor C1 discharges thru coil (RC constant determines nap duration)
3. C1 discharged =⇒ coil de-energized
4. Poles fall back to N.C. contacts
5. Pi powers on & reboots
6. Next test cycle begins

Cycle time ≈ 1 min

PiNap Circuit Prototype



Complete Testbed



Tips for Builders

- Use supercap instead of electrolytic for C1
- Mind the polarity of components
  - diode, relay coil, capacitors
- Avoid GPIO pins that fluctuate during boot
- See *CACM*/*Queue* article for details

Host System Configuration & Test Software

- Lots of details
- Basic idea:

  at boot, `cron` job runs scripts that start software under test, wait a while, trigger power-off via GPIO pins
- Make sure files don't pile up in places like `/var/tmp`
- Read the *CACM* or *Queue* article
- Tarball of code is provided:

  everything you need to reproduce my tests or run your own

Testing Persistent Memory

- Definitions
- Implementations
- Tests

Persistent Memory

- /= non-volatile memory
- *Software* abstraction
- Implementable on conventional hardware
  - ACM *Queue*, Vol. 17, No. 4, July/August 2019
  - USENIX *;login:*, Vol. 44, No. 4, Winter 2019
- Basic trick: lay out data structures in `mmap()`'d file
- Crash consistency: failure-atomic `msync()` (FAMS)

Is Failure-Atomic msync() Failure-Atomic?

- Let PiNap decide
- Test `famus_snap` library
  - very simple user-space FAMS
  - leverages file cloning: `ioctl(FICLONE)`
  - USENIX *;login:*, Vol. 44, No. 4, Winter 2019

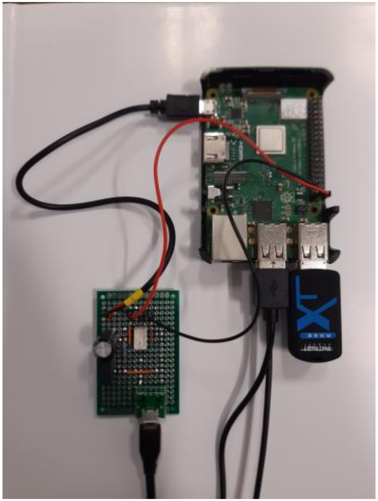
Test “Application”

- Repeatedly
  - Fill memory with pseudo-random pattern
  - Call failure-atomic `msync()`
- Cut & restore power with PiNap
- Check for corruption in backing file
- Three storage devices
  - $30 64-GiB flash thumbdrive
  - $90 500-GiB portable SSD
  - $220 512-GiB flash memory stick

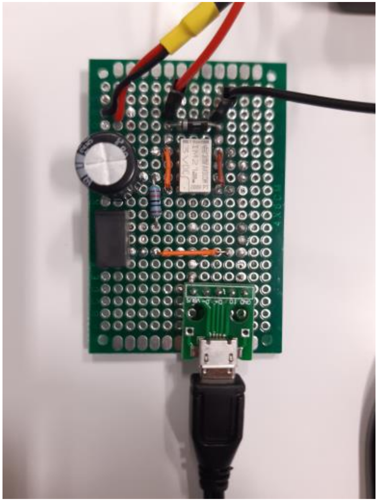
Results

- Over 58,000 power-off/on tests
- All tests pass
  - not one byte lost or damaged
- Caveat: Dijkstra on bugs

Second Testbed



Second Testbed



Using PiNap to Test Databases

- Goal: study SQLite relational database under power faults
- Methodology: Use `cron` job to:
  - load database with known initial data
  - run transaction workload to stress database and trigger errors
  - trigger power-off at a random time while running workload

Transaction Workloads

Two single-threaded transaction workloads (based on Zheng et al., OSDI 2014)

- Workload 1: Transaction updates multiple accounts with known values
    - tests large transactions
- Workload 2: Transaction moves money between accounts
    - tests multi-row consistency

Work in Progress

- Student is implementing SQL queries for above workloads
- Next step: first test SQLite, then test MariaDB (MySQL) and `mmap()`-based LightningDB

Summary

- Power failures threaten application data integrity
- Failure-atomic update mechanisms protect data
  - or so they claim
- Realistic testing is necessary
- RPi + novel PiNap circuit = cheap, useful testbed
  - under $100
  - 10,000 tests per week
- `famus_snap` library survives over 50K power failures
  - it's either reliable or very lucky
- Work in progress: Test SQLite relational database Work
- in progress (by others): Optane memory testbeds