Hypervisor-based Fault-tolerance
Bressoud and Schneider
SOSP 1995

Why are we reading this paper?
  We're about to look at a bunch of systems that replicate for fault-tolerance
  The hypervisor is at one extreme of the spectrum
  Two machines run the exact same instructions
  If one fails, the other just keeps going, no time/data/work is lost
  Totally transparent, runs existing O/S and apps
  Only a factor of two slower
  Seems like an amazingly general/easy/practical solution to fault-tolerance

Where they are coming from
  Banks, telephone exchanges, NASA need fault-tolerant computers
  1980s saw lots of *hardware* fault-tolerant machines
  Assumption: CPU most likely part to fail due to complexity
  So two CPUs connected to bus, single memory, disk, &c system
  All in the same cabinet. Not distributed over the Internet.
  Hypervisor is all s/w! Runs on stock h/w.

Let's design our own

Straw man:
  two identical machines
  start with same memory contents (program and data)
  just start them executing!
  if one fails, the other keeps going
  it's *fast* -- just as fast as ordinary single computer!
  are we done?

Q: Will they perform the same computation?
  theorem: yes, if they execute same operations,
    in the same order, and each operation has deterministic effect
  ADD instruction &c probably deterministic
  instructions to read time-of-day register, cycle counter, priv level
    not deterministic
  memory system?
  local devices like hard disk?
    low level: interrupt timing
    high level: whether disk read/write results are deterministic
    errored blocks...
  external input devices like network, keyboard?
    both CPUs need to get each input
  output devices like network, display, printer?
    exactly one CPU should produce each output

Let's figure out how to deal with local devices (e.g. hard disk)
  why not replicate all I/O hardware on each machine?
    e.g. let both machines read/write their own disk
    writes should be identical, thus reads also
  maybe some h/w replicas don't behave the same.
    for example, o/s knows how to re-map a failed disk sector,
      but the two disks won't fail at the same sectors
  so why did they replicate the main-memory system?

```
    doesn't fail? would be too slow to share?

So they hook up all devices to both machines: shared I/O bus
  sending commands and data to the device
    only the "primary" writes the device
    hypervisor ignores backup's I/O writes
  reading results back from the device
    only primary reads
    primary hypervisor copies to the backup
    backup hypervisor stalls until correct value arrives from primary
  handling interrupts
    only primary CPU sees real h/w interrupts
    hypervisor sends them to the backup

how to ensure two machines see interrupts at the same time?
  cannot let them occur at different points in app instruction stream
    that might cause different behavior
  could execute instructions one at a time on both prim and backup
    backup waits until it knows whether interrupt happened after prev
instr
    slow...
  epochs!
  CPU h/w can be told to interrupt every N instructions
    i.e. at exactly the same point on both machines
  primary hypervisor delays all interrupts until end of epoch
  backup has to wait at the end of its epoch
    maybe primary hasn't reached epoch end, hasn't seen all interrupts
  at each epoch end, primary and backup deliver all interrupts

what about fail-over?
  if primary fails, backup must start doing I/O
  how does backup know the primary has failed?
  what about I/O writes in the last epoch: might or might not have
happened.
  what if backup is waiting for an I/O read?
    did the primary even issue the read request?
    did the result come back, but not yet sent to backup?
  in the end, the O/S has to re-try the I/O!

do all devices follow the IO1/IO2 rules?
  hypervisor has to understand an operation was started by not finished
    so it can generate "uncertain" interrupt
  device has to support repeated operations correctly
  O/S device drivers have to know to re-try operations
  OK for disk read and disk write
    some SCSI drivers notice time-outs and re-try
  maybe OK for network (TCP will sort it out)
  not so OK for keyboard, line printer, ATM bill vendor

how does the hypervisor know about incoming I/O data from devices?
  e.g. disk blocks just DMA's from disk
  needs to send block data to the secondary
  no problem for programmed I/O (hypervisor can trap on LD)
  puzzle: DMA. how does the hypervisor know?

is the machine virtualizable?
  we can always interpret SLOWLY
```

```
  real question: can I run o/s and apps on the real hardware?
  will the hardware trap to hypervisor on *every* tricky instruction?
  i.e. any instruction that might be non-deterministic
    OR reveal the existence of the hypervisor?
  time-of-day register
  memory-mapped I/O loads and stores
    need to trap and substitute load values
  HP branch-and-link instruction
  HP TLB replacement
  HP "space registers" (page mapping tables?) writable by users (!)

What if ethernet cable breaks?
  primary still running
  backup will promote itself
  that's part of what the fail-stop assumption assumes away...

What if we want to re-start a failed+repaired primary?

Is this the way we should build all fault-tolerant systems?
```