High Availability

1. Questions from reviews:
    a. How does recovery start after a failure?
        i. Boot up VM from snapshot
    b. Flat curve with increase in checkpoints for network buffers?
        i. Not snapshotting at desired frequency because snapshot takes too long, so no change
    c. What happens after failure?
        i. Need to recopy entire VM – full snapshot – to fully repair
    d. How does checkpoint frequency relate to fault tolerance?
        i. It doesn't; it relates to network latency
        ii. Higher frequency checkpoints -> lower latency but higher overhead
    e. How do clients move to backup?
2. Goals
    a. High availability for what failures?
    b. Unmodified applications
3. Commercial high-availability systems
    a. Vendors:
        i. Tandem, stratus
        ii. IBM, HP
    b. How built?
        i. Special purpose hardware
            1. Dual redundant processors with lockstep execution
            2. Redundant cross-over networks
            3. Dual-path storage
    c. Where used?
        i. Banks, etc.
4. Cloud-based high availability systems
    a. Platform:
        i. Commodity HW, OS
    b. Infrastructure:
        i. Redundant networks
        ii. Network storage – GFS
        iii. Redundant HW – store things multiple times
    c. Software
        i. Written to distribute requests automatically, detect failure, retry/recovery quickly
        ii. Everything custom:

            1. Client apps detect failure, know about replicas and try other replicas
            2. Services know about failure, try other services. Know about storage replicas, try other storage replicas
5. Hypervisor-based fault tolerance:
    a. General idea;
        i. Take non fault-tolerant code, put a layer under it that replicates it transparently
        ii. Question: what failures can be tolerated?
            1. Applications? OS? Hypervisor? **Hardware**?
        iii. Compare to application-level replication
    b. Idea 0:
        i. Run in a hypervisor on shared storage
        ii. If crash, restart somewhere else from shared storage
            1. Just like local reboot, but could be done faster
            2. But lose data during crash
    c. Idea 1:
        i. Feed all inputs from one system to another
        ii. Should lead to duplicate states
        iii. Challenge: non-determinism
            1. Interrupts delivered at different times
            2. Timestamps on events (e.g. http requests) vary
            3. Expensive to fix
    d. Idea 2:
        i. Replicate complete system state from one hypervisor to another
        ii. Block output until replication completes
            1. Avoid producing an output that could be lost
            2. No externally visible state should be lost
                a. Example: report data saved, but is not saved
    e. Challenges:
        i. Good performance:
            1. Need to replicate memory state
            2. Can't release output until memory has been replicated
            3. Could cause lots of delays if synchronous
                a. Do op ; replicate ; get ack ; release
        ii. Data volume
            1. Often cheaper to ship an operation than the data
                a. E.g. adding value to a hashtable touches many pages but the key/value may be small
                b. Reorganization (e.g. btree, rehashing) lead to lots of data changes from small operations
6. Remus design:

a. Overview:
    i. Run a primary
    ii. Periodically snapshot and send snapshot to backup
    iii. Delay output at primary from before snapshot until snapshot arrives at backup
        1. But keep executing ahead
    iv. Storage:
        1. Disk writes propagated immediately to backup where buffered until RAM snapshot arrives
    v. Backup does not execute – is just state in memory/disk – until primary fails
b. Failure model:
    i. Keep running with single machine (hardware) failure
    ii. Reboot from dual failure (like a normal crash)
c. Xen terminology
    i. Architecture:
        1. Hypervisor
        2. Dom0 – management code, device drivers
        3. DomU – guest VM
    ii. XenStore – centralized config database, place to share data between VMs
    iii. XenBus – bus abstraction for drivers in guests to talk to to other VMs
    iv. XenD – management Daemon in Dom0, starts/stops/creates VMs via hypercalls to Xen
    v. Dom0

7. Remus implementation
    a. Leverage existing live migration:
        i. Migrate running VM to another machine
            1. Not start machine at destination
            2. Continue running at source
    b. Fast snapshots/checkpoints
        i. Divide time into epochs between snapshots
        ii. Once per epoch, pause running VM & copy changed state into buffer
        iii. Transport buffer to backup
        iv. Ack backup to primary
        v. Release output
    c. Memory/Cpu snapshot
        i. While running epoch, track all modified pages
        ii. At end, mark all those read-only, copy to backup, then make writable

    iii. Mark memory read-only, copy dirty pages, make writable
        1. Do in the VMM, not guest
        2. Can track all pages modified since previous epoch
    iv. Repeat until # of pages dirtied during copy == # of pages copied
        1. Initially lots of dirty pages
        2. When not converging, pause VM and copy remaining dirty pages
    v. Implementation details:
        1. Optimize communication path to guest to tell it to suspend for final stop-and-copy
        2. Map guest physical pages into a process in management VM completely to do copy to avoid lots of map/unmap operations
        3. Copy modified pages to staging buffer to allow immediate execution; can restart VM before passing pages along
d. Buffering output
    i. Why buffer output until checkpoint complete?
        1. If not, may announce something happened, when backup cannot (or will not) do that
            a. Example: receiving email; could ack. Was received but then would get lost if not replicated before backup
    ii. Implementation:
        1. Use network queueing discipline in VMM: block outbound packets until receive a release essage
        2. Copy off shared ring buffer for greater buffering space
e. Disk buffering
    i. Why different than network?
        1. Network can lose, reorder packets
        2. Need to recover contents on dual failure (goal of system)
    ii. Solution:
        1. Mirror disk contents completely to backup
        2. While running, writes to disk tracked and checkpointed
            a. Writes are write-through: go to local disk + backup memory
            b. Ensures primary doesn't go to fast due to local disk writes
                i. Otherwise if disk writes only on backup, primary gets ahead and backup cannot catch up
        3. Backup writes out blocks after receiving memory state off following checkpoint

            a. Alternate writing primary & backup
            b. On double machine failure, One is always most recent and correct (one not being written)
    f. Recovery:
        i. Detect failure via heartbeat
        ii. Start VM on backup (load VCPU registers into real CPU, start running)
        iii. Move clients to new machine
            1. Done at switch: send reverse ARP saying an IP address now has a new Ethernet address
            2. A few packets get lost in the middle while original machine isn't responding
    g. Repair
        i. Eventually fix primary (or backup)
        ii. Need to re-replicate potentially everything (all of memory, all of virtual disk)
        iii. Then can be fault tolerant again.
8. Fit into fault tolerance framework:
    a. Fault detection: heartbeats
    b. Isolation: separate VMs
    c. Recovery: backwards to last checkpoint at backup
9. Evaluation
    a. Question: what should be evaluated?
        i. Reliability: how?
        ii. Performance: what are considerations?
            1. App performance
                a. Throughput – hurt by overhead
                b. Latency of requests – hurt by waiting for replication to complete
            2. Microbenchmark: determine what affects performance
                a. Look at amount of data written to see how affects copy time
                b. Look at frequency of checkpoints to see how affects performance
10. Sources of inefficiency
    a. Copies entire page when partial page modified
        i. Not evaluate ratio of pages copied to size of requests
        ii. Solution: compression/diff
    b. More pages dirtied means slower checkpoints means more overhead
        i. Better to checkpoint more often when fewer pages dirtied
        ii. Can slow down VM if dirtying pages too much to keep checkpoint overhead low

c. Copy on write
    i. Remus copies all dirty pages synchronously at snapshot (pausing VM
    ii. Could mark read-only, copy slowly

11. Big design issues:
    a. Requires 1 hot backup per server
        i. May require double capacity to tolerate failures, as have to have idle spare that is busy for every machine
        ii. Do not evaluate how many different VMs can be backed up from a single server at once
            1. E.g. 5 VMs backed up to 5 different places or one place?
            2. Can a single machine server as a backup for 5 other machines?
        iii.