Lecture 16 CS 695

	Average	Median	Max.	S.D
Quiz1 (18)	8.23	8	16.5	3.9
⊛ Midsem (30)	13	12.8	27.25	
A1 (15)	6.8	7	15	
⊛ A2 (15)	12.6	14	15	

[remus] [snowflock]

↳ High Availability via Asynchronous VM Application

~ HA : highly available
- ↳ service is available (for use/access) at all/most times.
- ↳ TTR ~ (time to recovery (restart)) small
- ↳ failure recovery model.
  - ↳ relate to state exposure
    - what is visible to the external world?
      - network
      - disk/storage (IO)
    - crash-consistent state.

⊛ crash-consistency

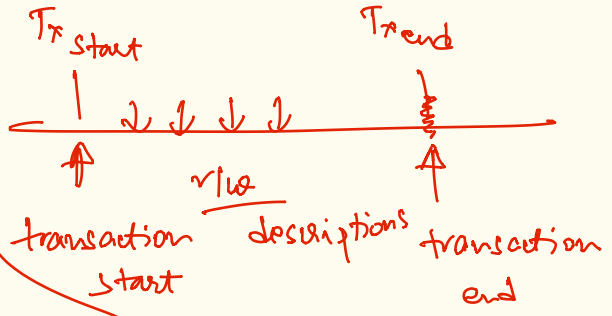
e.g: ① file system of native OS. (bare-metal)

fd = fopen("helloworld.txt");

↳ open — fd table — fs struct — inode — dentry

update blocks — read blocks from disk

② journaling ⟶



③ application-centric model

- HA in-line w/ application logic/code.

(appl'n's that are critical rich / scarad)

④ remus ⟶ system-wide replication (appl'n-agnostic)

- generalizability ~ all apps
- transparent ~ seamless in usage
- consistent failure model. ~ HA-as-a-service
  HA for the masses.

---

* system-wide replication.

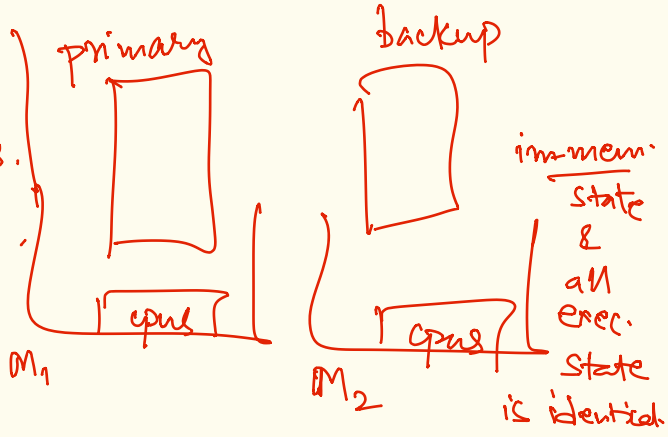
- in-memory state of the system has replicas.

runtime execution context that is ready to go on failure.

soln. trajectories.

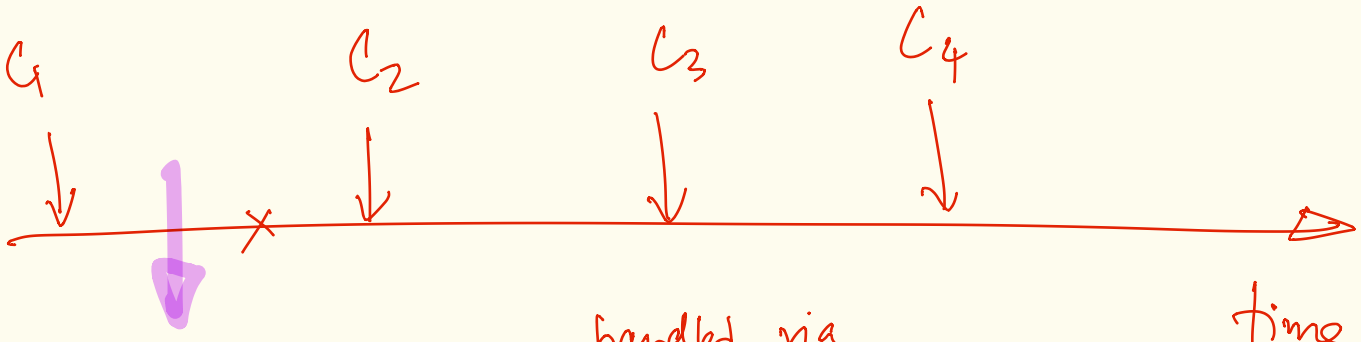
(i) reproduce input on multiple instances.

- instruction replay & IO event injection.
- record & replay ~



(ii) memory snapshots / checkpoints

- similar to live migration — service for all VMs.
  - copies memory state across machines (primary & backup)
    - has an active (primary) & a passive (backup) instance → multiple VMs



visible state (network & disk) ~ handled via speculative execution.

1. speculative (normal) execution — buffer all external state — n/w pkts, disk IO
2. checkpoint
3. transfer checkpoint
4. ack chkpoint & release externally visible state.
5. start next epoch.

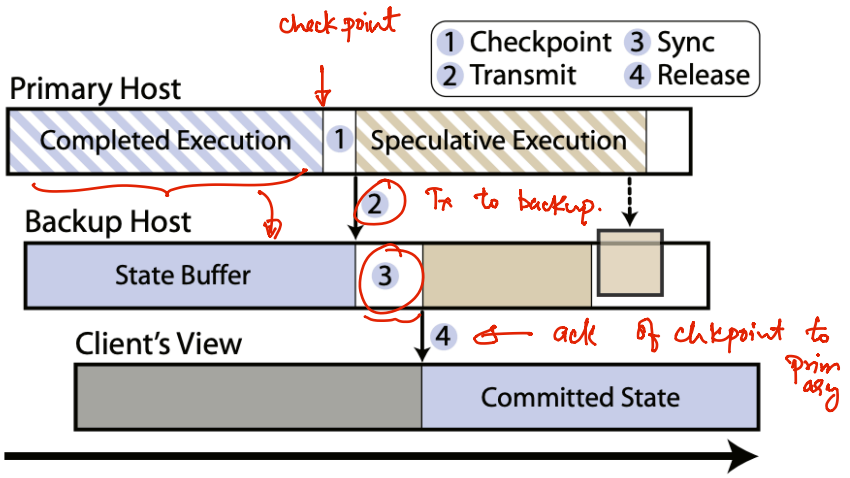


Figure 1: Speculative execution and asynchronous replication in Remus.

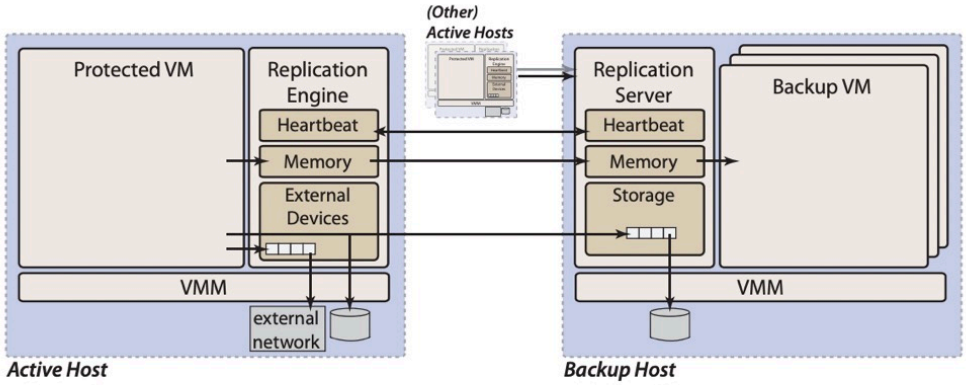


Figure 2: Remus: High-Level Architecture

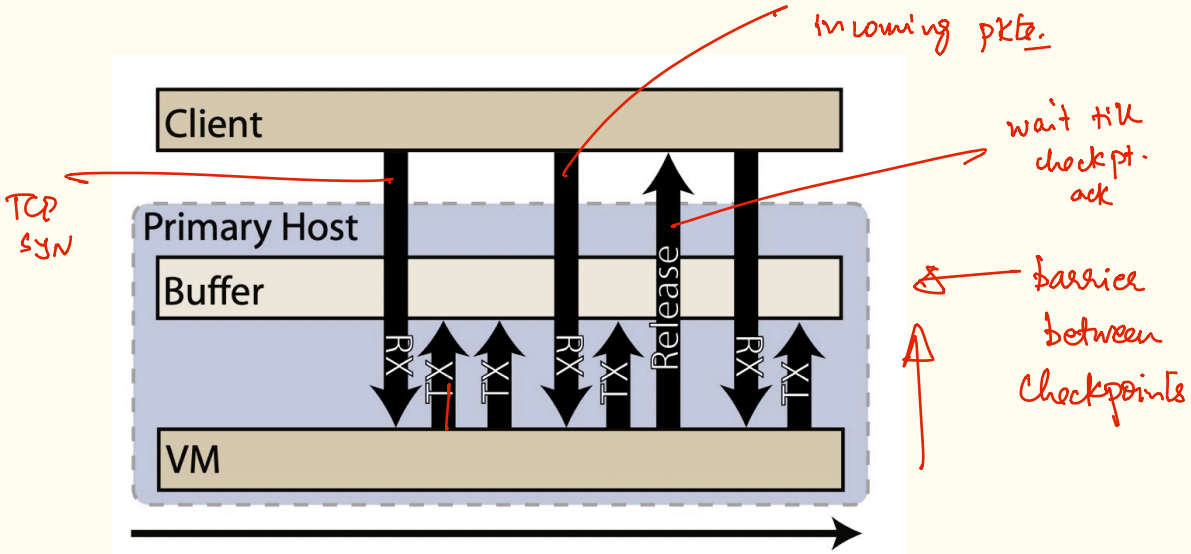


Figure 3: Network buffering in Remus.

disk IO:

different
- failure domains
for disk
- each VM has
its own local
disk.

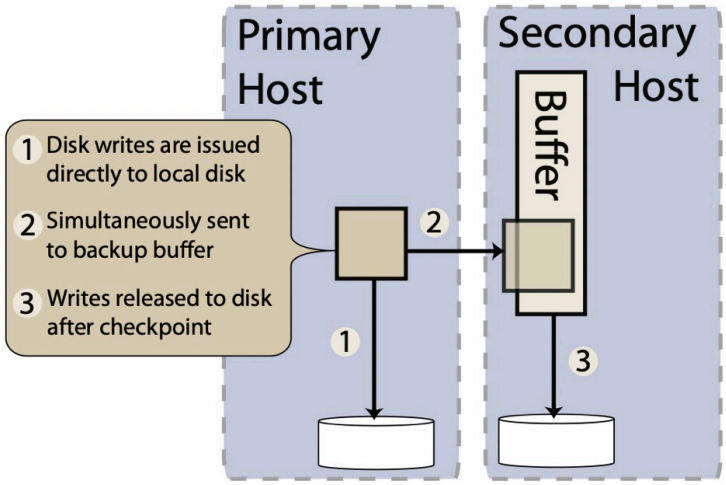


Figure 4: Disk write buffering in Remus.

[Snowflock]

(a) Sandboxing

```
state = trusted_code()
ID = VM_fork(1)
if ID.isChild():
 untrusted_code(state)
 VM_exit()
else:
 VM_wait(ID)
```

(b) Parallel Computation

```
ID = VM_fork(N)
if ID.isChild():
 parallel_work(data[ID])
 VM_exit()
else:
 VM_wait(ID)
```

(c) Load Handling

```
while(1):
 if load.isHigh():
  ID = VM_fork(1)
  if ID.isChild():
   while(1):
    accept_work()
 elif load.isLow():
  VM_kill(randomID)
```

(d) Opportunistic Job

```
while(1):
 N = available_slots()
 ID = VM_fork(N)
 if ID.isChild():
  work_a_little(data[ID])
  VM_exit()
 VM_wait(ID)
```

Figure 1: Four programming patterns based on fork's stateful cloning. Forked VMs use data structures initialized by the parent, such as `data` in case (b). Note the implicit fork semantics of instantaneous clone creation.