Speculative Recovery:

Cheap, Highly Available Fault Tolerance with Disaggregated Storage

USENIX ATC'22

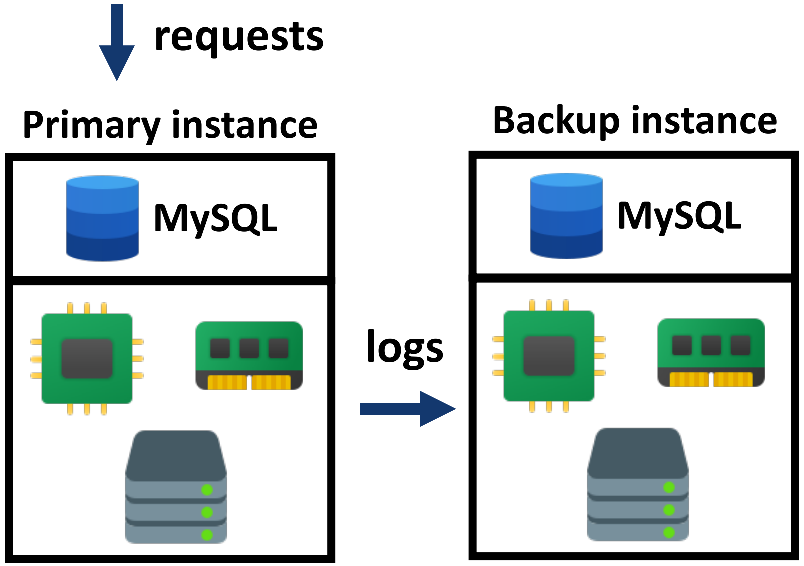
2022.05.24

Background

Application Fault-tolerance

Traditional technique - **Application-level Replication**

- Replicate the application across multiple compute instances
- Drawbacks:
  - ➢ Costly
  - ➢ Separate implementation

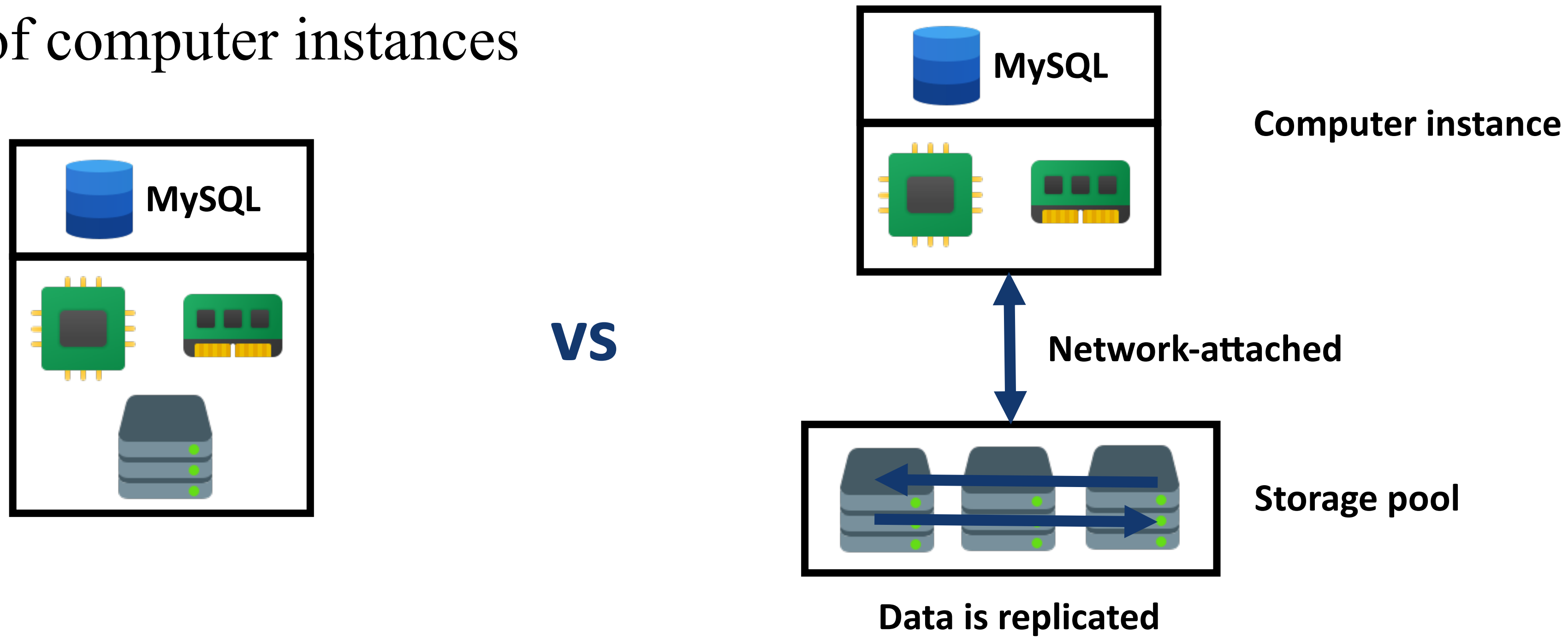


Background

Recovery From Disaggregated Storage (REDS)

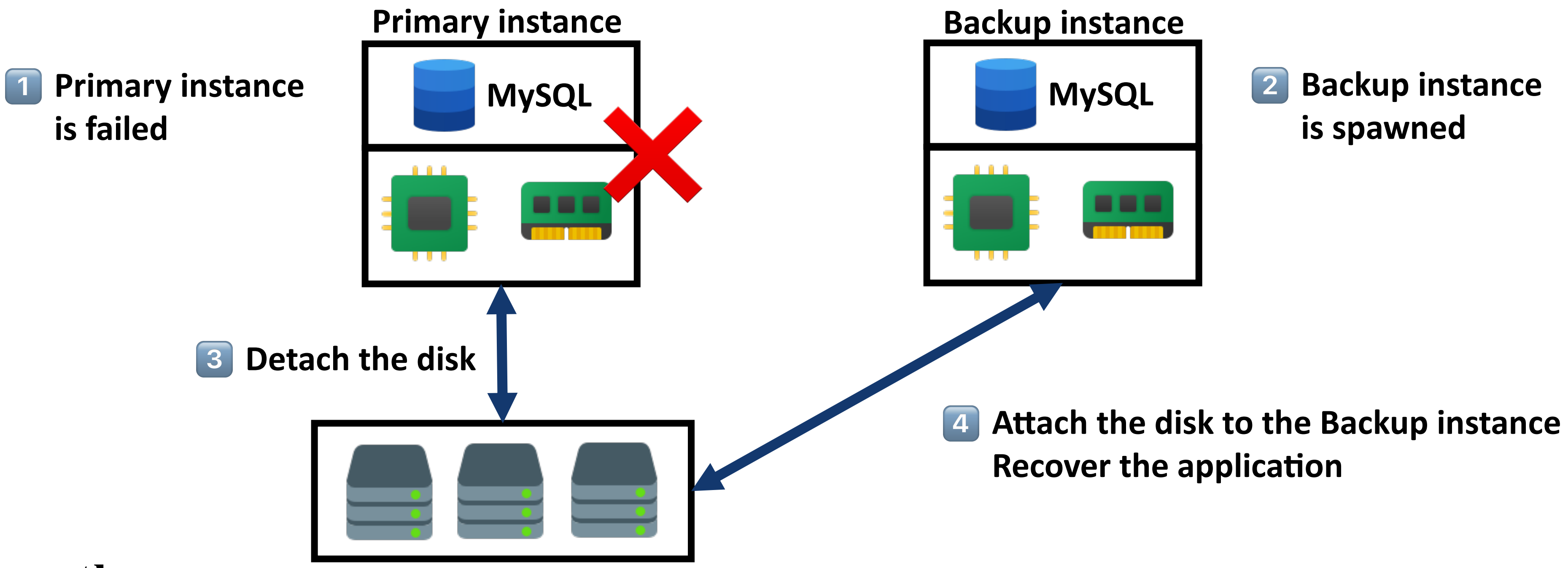
Opportunities

- Disaggregated storage
- Fast provision of computer instances



Background

Recovery From Disaggregated Storage (REDS)



Strengths

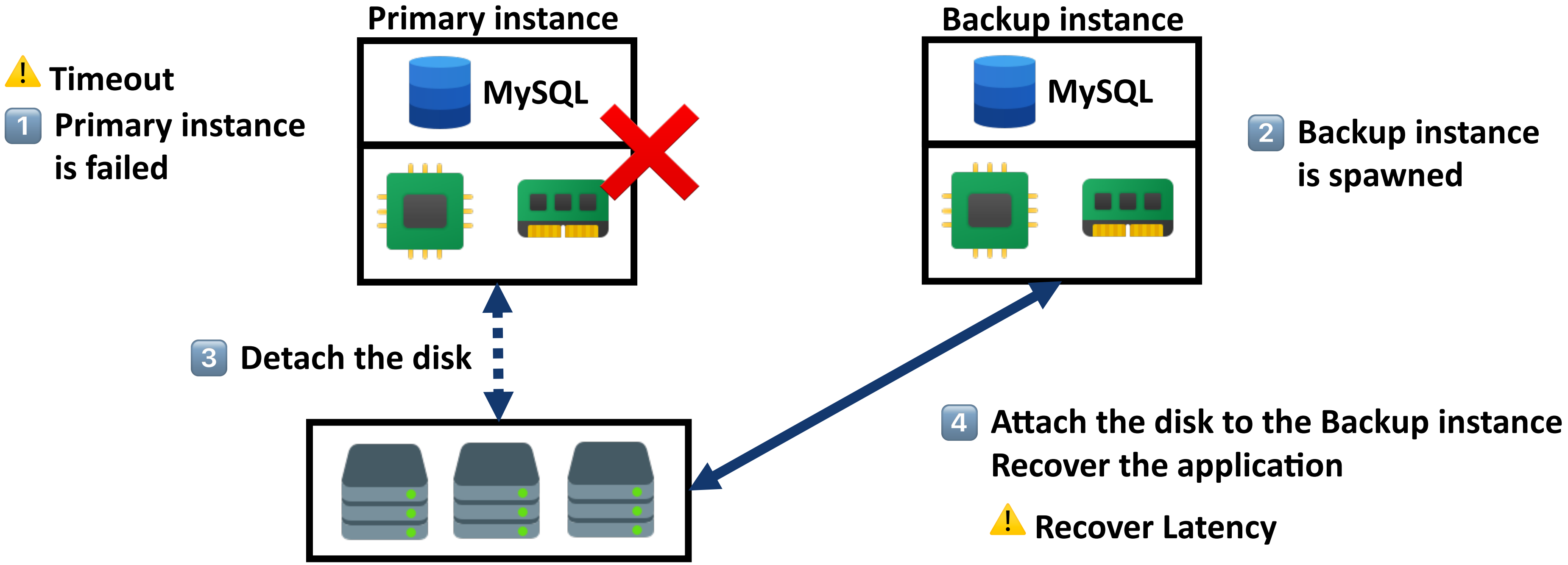
- ➢ Low cost
- ➢ Generally applicable to crash-consistent applications

Background

Recovery From Disaggregated Storage (REDS)

Drawback – Low availability

- Failover must be sequential
- The *future* is unknown

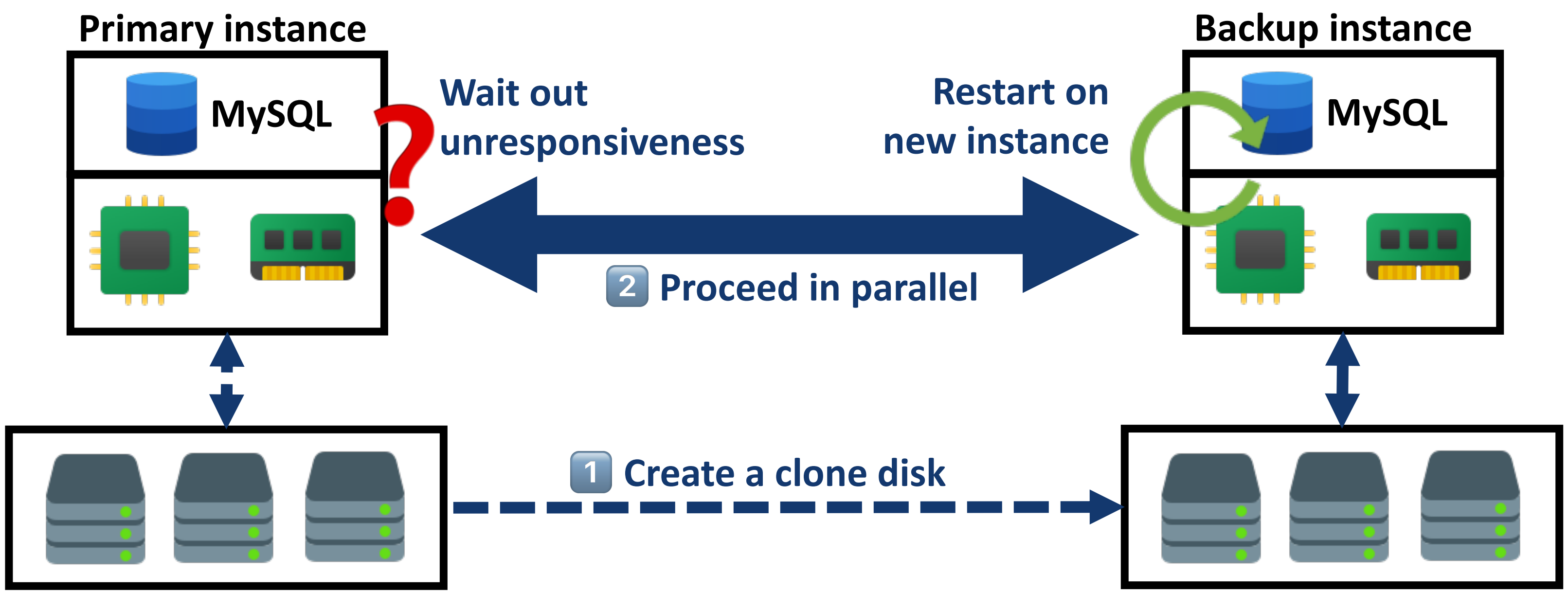


Main Idea

Speculative Recovery

- On *primary* downtime detection, *backup* initiates recovery from cloned disk, possibly **concurrently** if primary still up.

Aim: Increase the **availability** of apps that achieve cheap fault tolerance using **REDS**



Main Idea

Speculative Recovery

Challenges

- How to ensure application **correctness** ?
- How to ensure good disk **performance** for the backup instance to recover the application ?

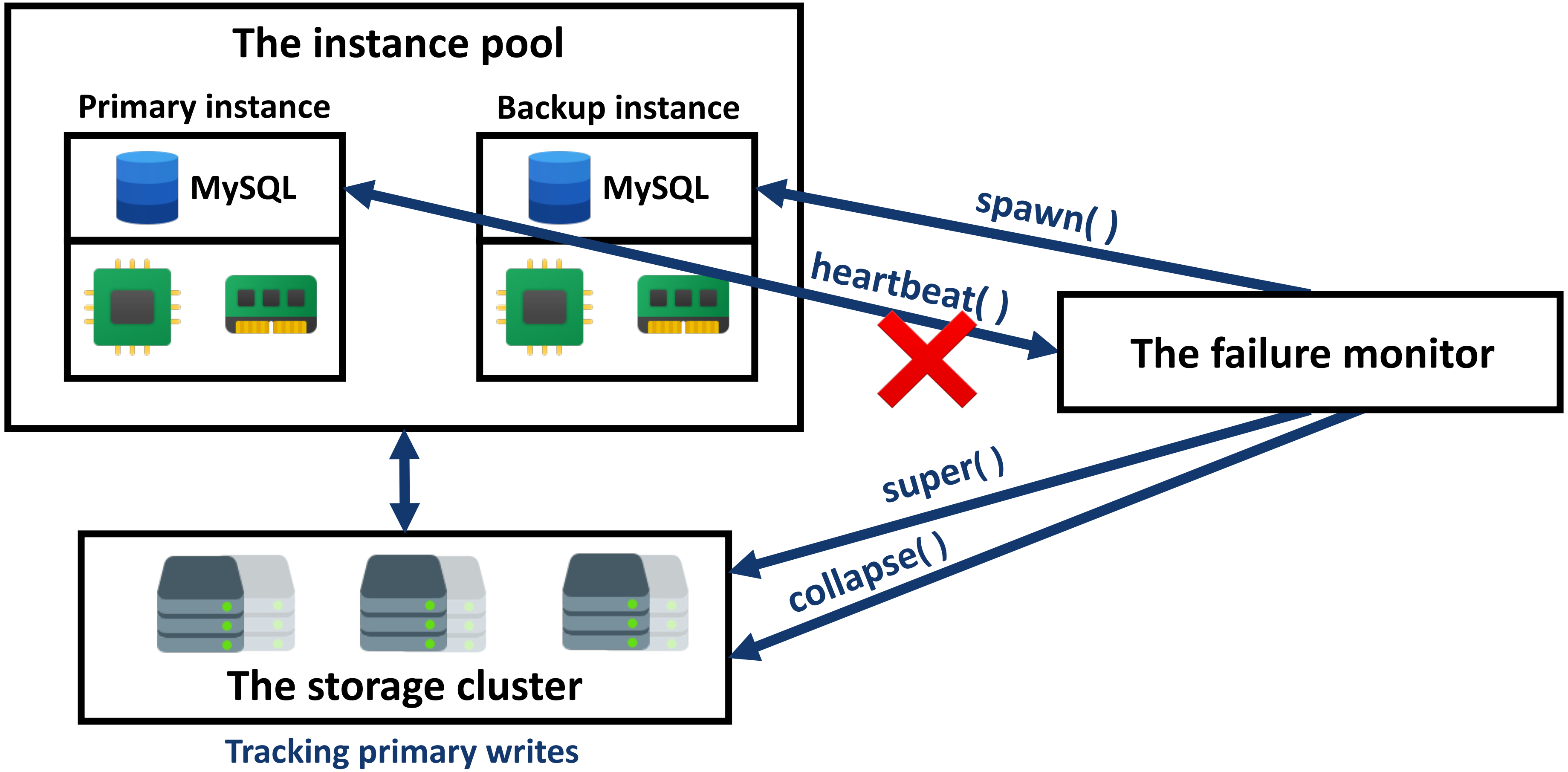
Key Designs

Introduce two new disaggregated storage primitives:

- ***super*** : creating a superposition by creating a disk clone
- ***collapse*** : collapsing the superposition by tracking writes to the primary's disk

Architecture Overview

Speculative Recovery System



Key Design 1: Creating a Disk Superposition - *super*

Copy-On-Write (COW)

Existing designs for COW disk clones perform very **poorly** for recovery workloads

- Copy dirtied blocks to different storage shards

  Result in considerable overhead
- Each dirtied block requires a blocking operation to allocate a new location in the storage area network

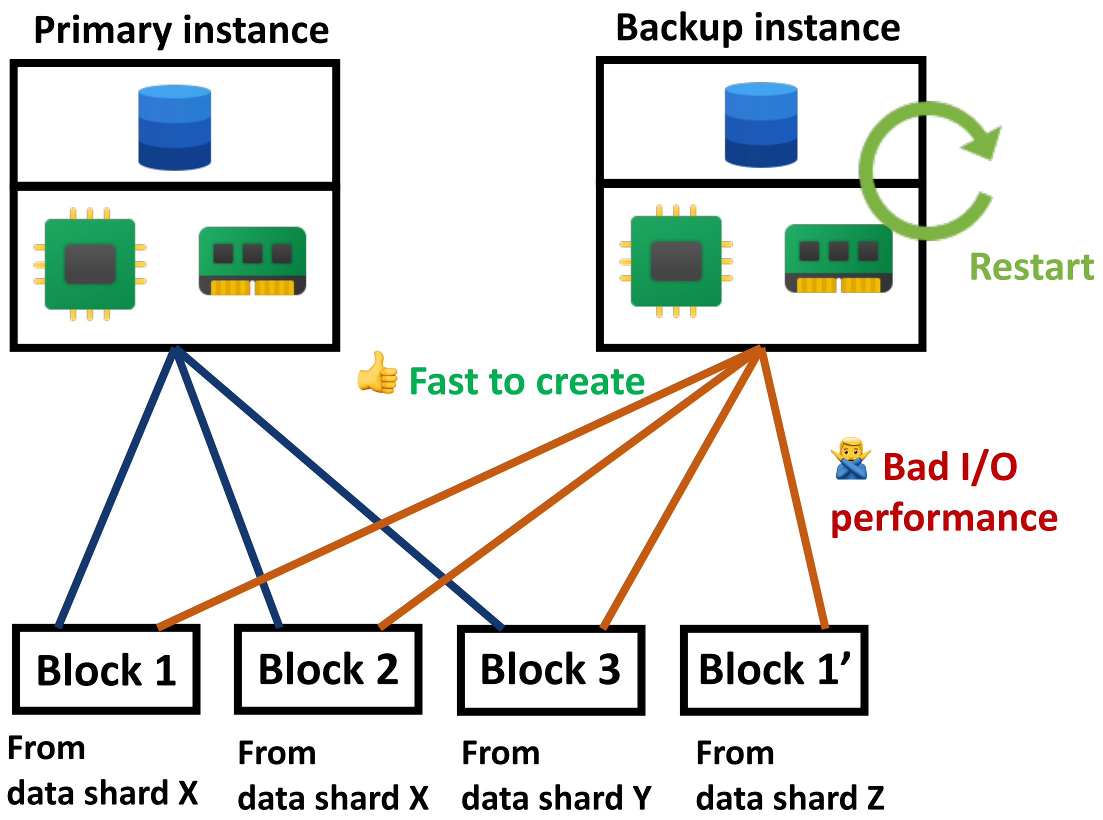
  Eliminate parallelism benefit for concurrent writes

Parent Disk allocation table

Block 1 : shard X
Block 2 : shard X
Block 3 : shard Y
…

Child Disk allocation table

Block 1' : shard Z
Block 2 : shard X
Block 3 : shard Y
…



Key Design 1: Creating a Disk Superposition - *super*

Collocated-Clone

Reuse parent's allocation table to collocate child blocks with their corresponding parent blocks.

- No need to traverse the network again when copying a dirtied block
- Never require a blocking allocation operation

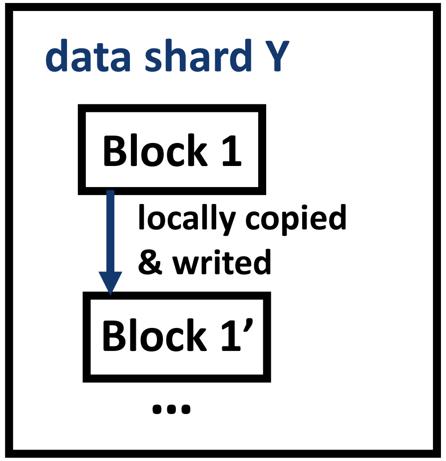
Parent Disk allocation table

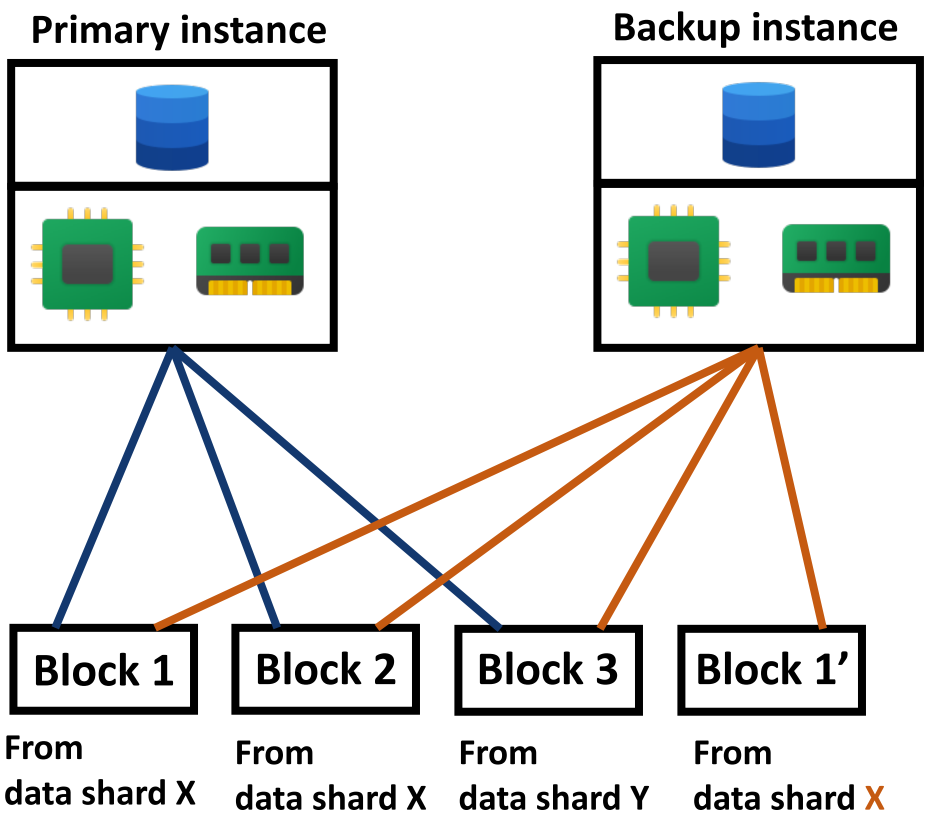
Block 1 : shard X

Block 2 : shard X

Block 3 : shard Y

…





Key Design 2: Collapsing a Superposition - *collapse*

Problem: Letting parent and child disks diverge in superposition introduces potential app inconsistency, which must be hidden from clients

- ***dirty*** bit: whether writes have been applied to the parent disk since the creation of the child.
- ***allow-write*** bit: whether have permission to write on the parent disk

} tracking shard

Tracking *primary* writes: when *super*, before *collapse*

- Set *dirty* bit ← 0, *allow-write* bit ← 1
- When a shard of the parent disk **receives a write request**: Set *dirty* bit = 1

Atomic promotion: when *collapse*

- Check dirty bit:
  - 0 : deallocating the parent disk, proceeding with promotion of *backup*, setting *allow-write* bit ← 0
  - 1 : deallocating the child disk, aborting recovery

Experimental Setup

Implement a prototype speculative recovery system: **SpecREDS**

- Based on Ceph's block device interface ***rbd***

Compare 3 disk types

- ***rbd*** (a regular disk)
- ***rbd-clone*** (with Ceph's existing clone implementation)
- ***super*** (with collocated-clone)

Compare 3 systems

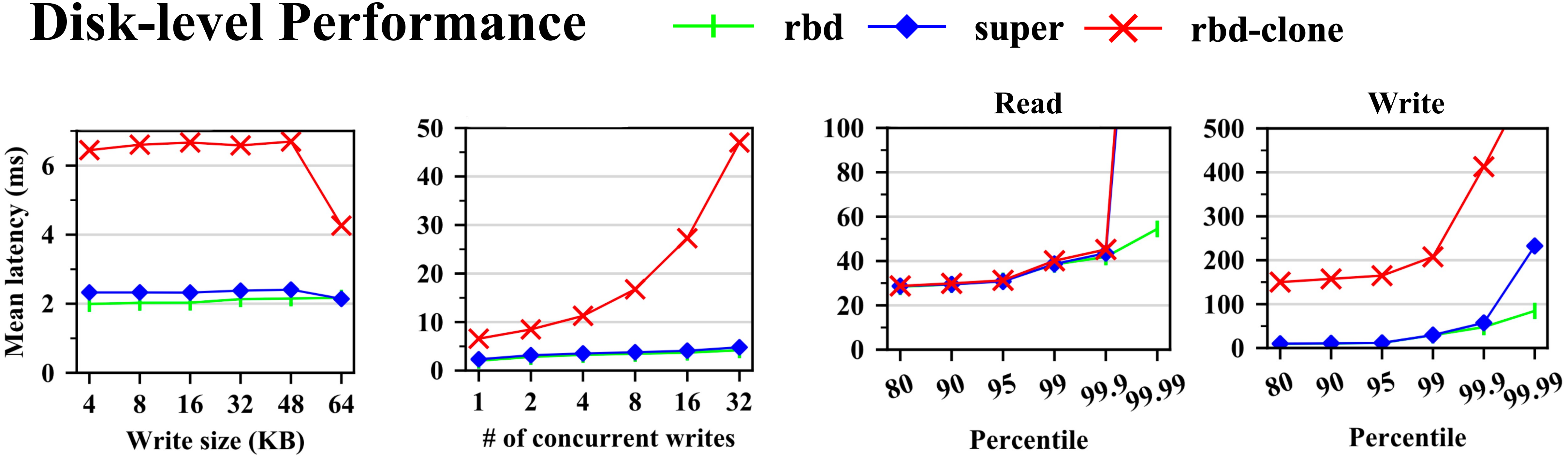
- ***REDS*** (using rbd)
- ***SpecREDS*** (using super)
- ***Oracle*** (using rbd)

Testing 3 database applications

- ***MySQL*** (with InnoDB)
- ***PostgreSQL***
- ***MariaDB*** (with RocksDB)

Evaluation

Disk-level Performance



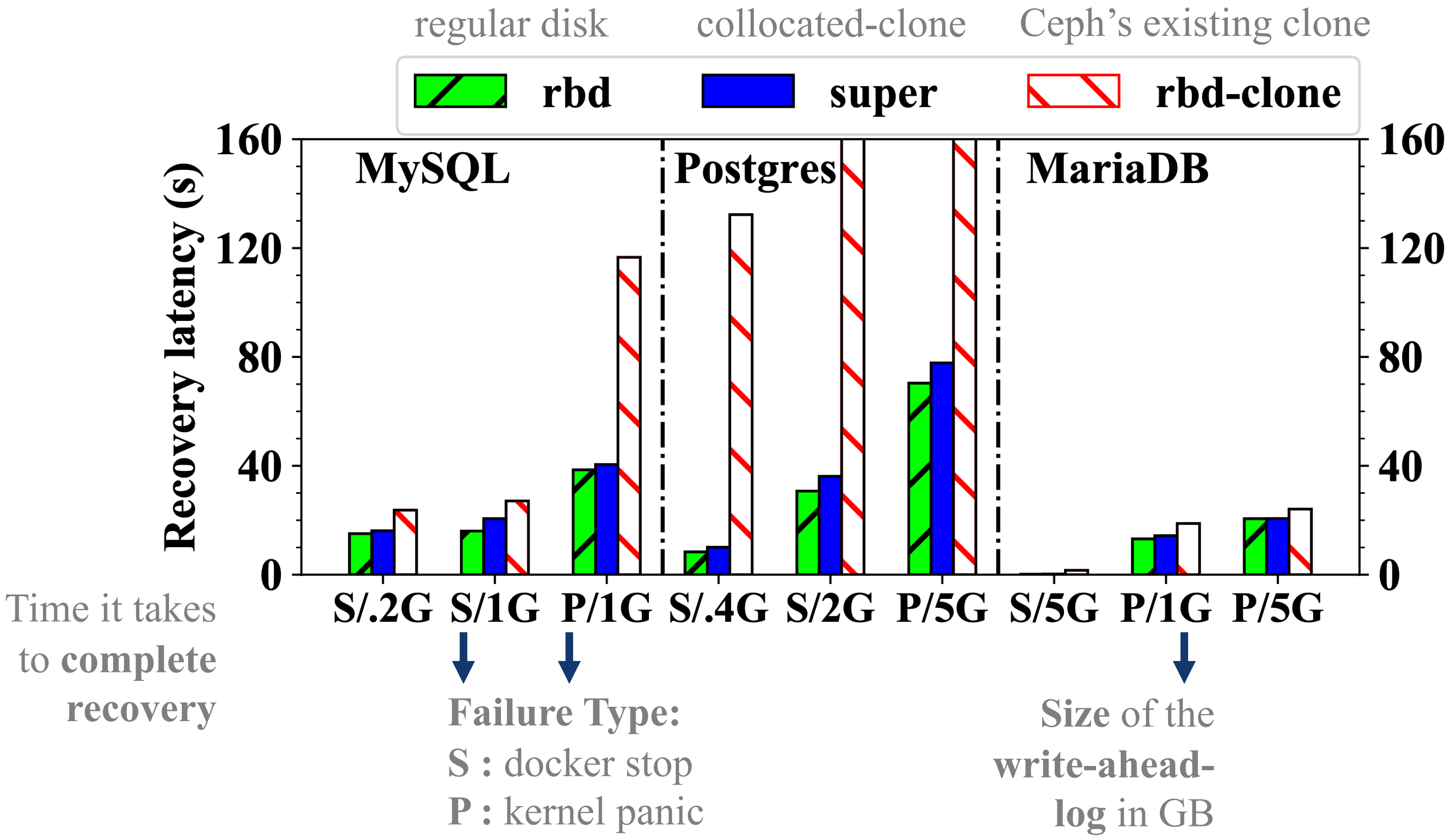
Single COW write latency | Concurrent COW writes | Performance on real recovery workloads

The disk-level improvements of super can achieve **recovery latency very close to** a regular rbd disk in real failure scenarios

Evaluation

Application Recovery Latency

- *super* **improves performance** over *rbd-clone*
- Recovery on *super* is only **slightly slower** than recovery on *rbd* by 13% on average.

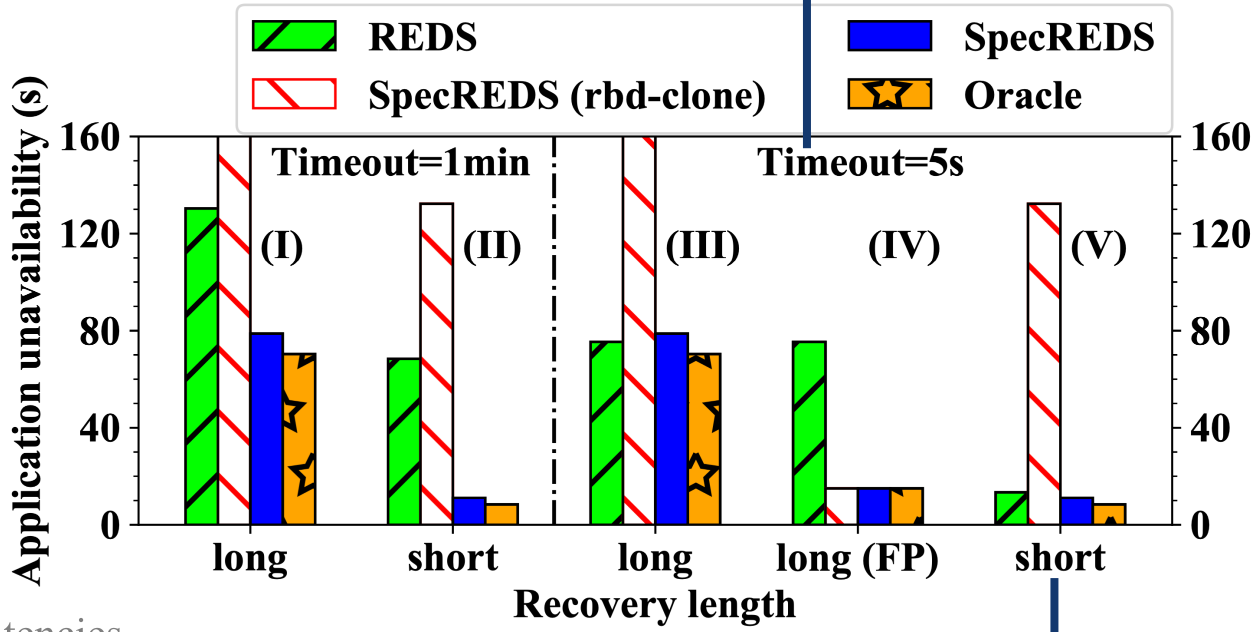


Evaluation

End-to-end Failover Latency

- the failover latency of SpecREDS (*rbd-clone*) is consistently the **highest**
- SpecREDS achieves significantly **lower failover latency** when REDS uses a medium timeout or for FP when REDS uses a short timeout
- SpecREDS is always **close to** the oracle lower bound

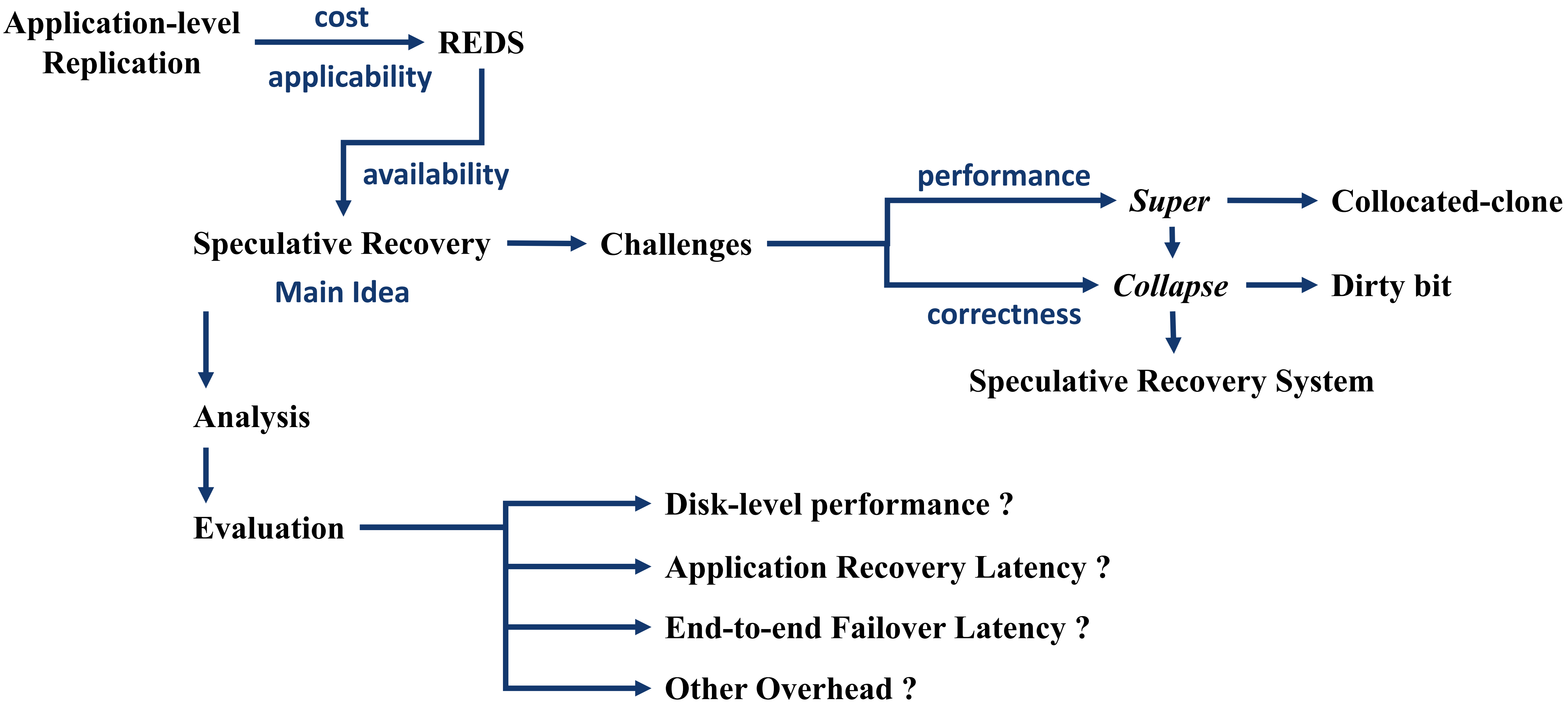
Long timeout : ~ **1** min
Short timeout : ~ **5** s



adds latencies together

Long recovery : ~ **1** min
Short recovery : ~ **5** s

Paper Summary



About

Why choose

- Knowledge of fault tolerance techniques (REDS)
- Improvements to cloud-edge storage
  - ✅ File recovery
  - ❓ Application recovery