30th ACM Conference on Computer and Communications Security (CCS 2023)

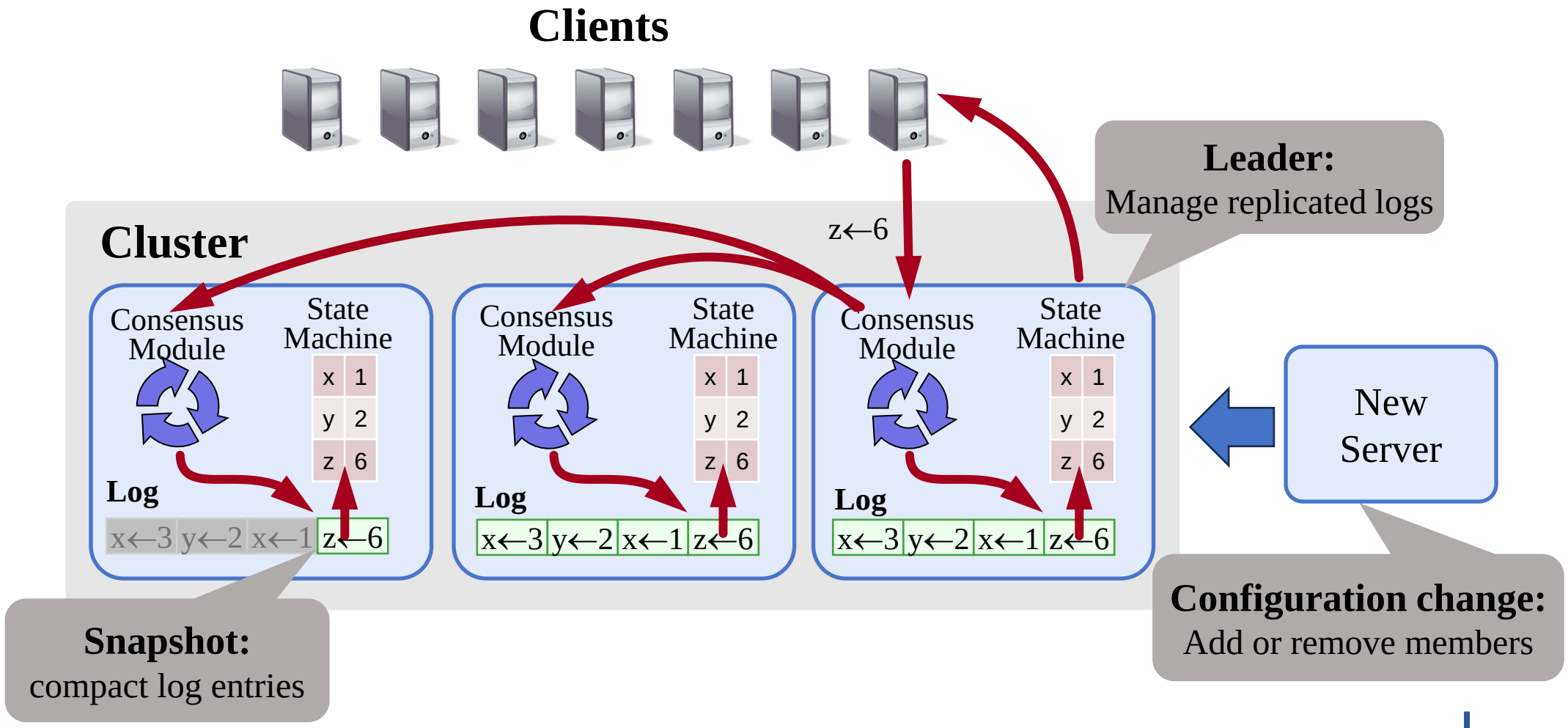
Greybox Fuzzing of Distributed Systems

Ruijie Meng
National University of Singapore
ruijie@comp.nus.edu.sg

Co-authors: George Pîrlea, Abhik Roychoudhury and Ilya Sergey

Bug in Distributed Systems

- Workflow of the Raft consensus protocol:

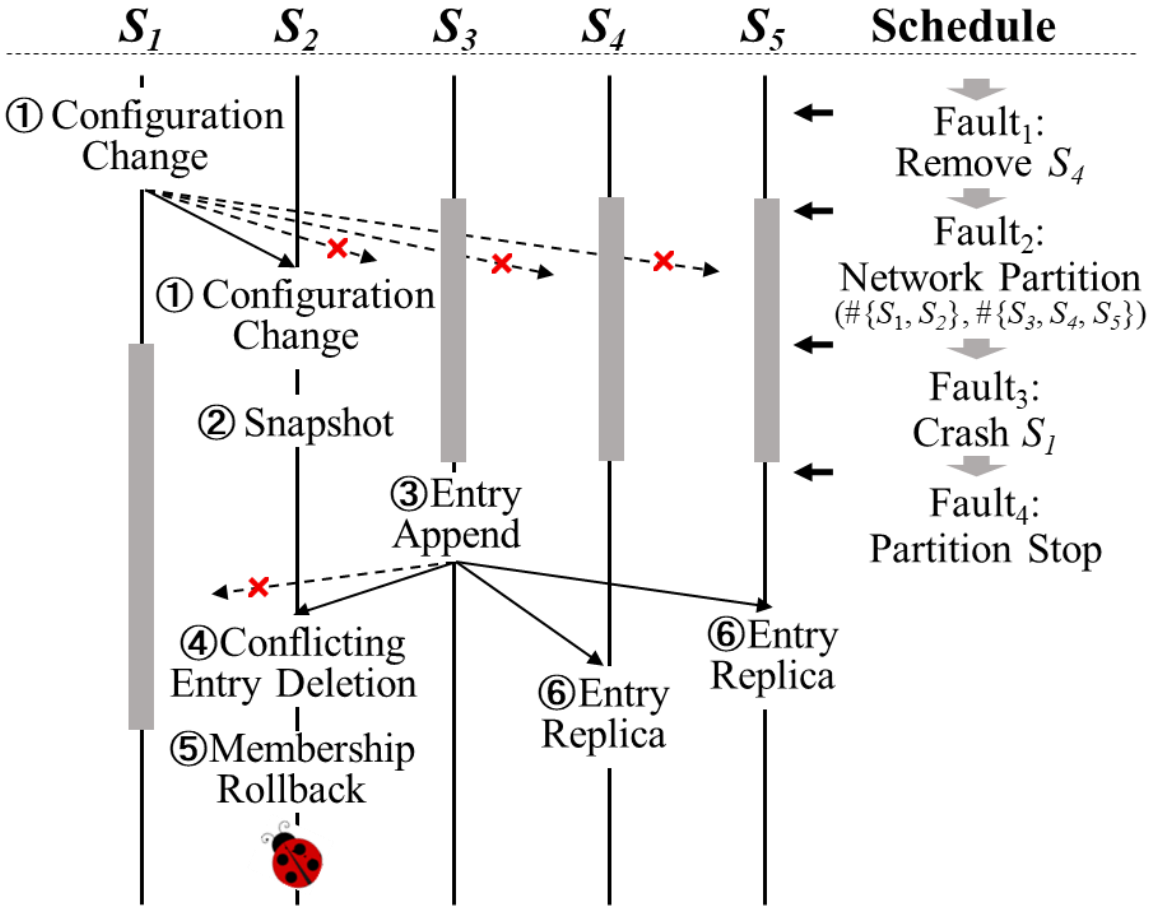


Bug in Distributed Systems

- Membership rollback bug in Canonical Dqlite:

```
145   int membershipRollback(struct raft *r){
146     ...
158     // Fetch the last committed configuration entry
159     entry = logGet(&r->log, r->config_index);
160     assert(entry != NULL);
176   }

986   static int deleteConflictingEntries(){
987     ...
1007    // Possibly discard uncommitted config changes
1008    if (uncommitted_config_index >= entry_index){
1009      rv = membershipRollback(r);
1010    }
1042  }
```



Testing Distributed Systems

Systematic testing → whitebox fuzzing

- ✓ Exercise complex event interleavings to find "deep" bugs
- ✗ Heavyweight: require a manually-written pervasive test harness or a system-level interposition layer
- ✗ State explosion: not able to scale to large systems

Stress testing (e.g., Jepsen) → blackbox fuzzing

- ✓ Low cost of adoption
- ✓ commendable scalability
- ✗ Ineffective to reach deep program behaviors

Greybox Fuzzing?

Can we find a balance between ease-of-use and effectiveness??

But there is no greybox fuzzing for distributed systems.

We explore this opportunity to extend Jepsen with feedback guidance from the program behaviors

Conventional Greybox Fuzzing

☛ We need to consider three questions while greybox fuzzing distributed systems:

Q1: What is the input space to distributed systems that could be explored adaptively?

A1: Schedules to inject faults (e.g., network partition)

Algorithm 1 Coverage-based Greybox Fuzzing

Input: Seed Inputs S

```
1:  T✗ = ∅
2:  T = S
3:  if T = ∅ then
4:      add empty file to T
5:  end if
6:  repeat
7:      t = chooseNext(T)
8:      p = assignEnergy(t)
9:      for i from 1 to p do
10:         t' = mutate_input(t)
11:         if t' crashes then
12:             add t' to T✗
13:         else if isInteresting(t') then
14:             add t' to T
15:         end if
16:     end for
17: until timeout reached or abort-signal
```

Output: Crashing Inputs $T_✗$

Q3: How to mutate inputs?

A3: Incrementally select action by action to construct a new schedule via Q-learning

Q2: What can represent program behaviors of distributed systems?

A2: Lamport timelines that is analogues to code paths

Greybox Fuzzing of Distributed Systems

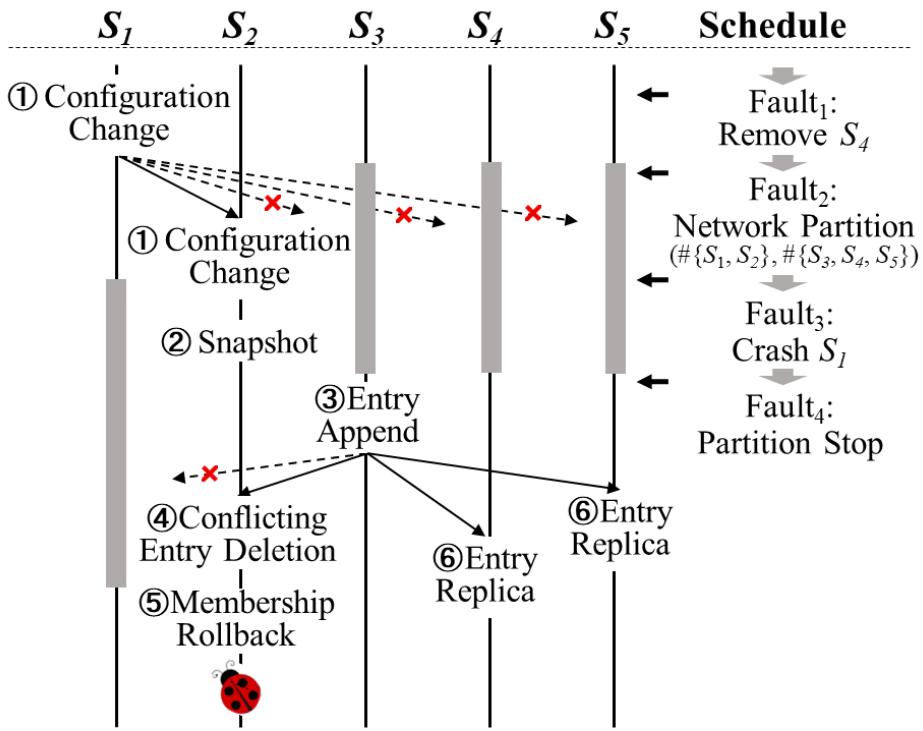
Algorithm 1: Greybox Fuzzing of Distributed Systems

Input: P_0: system under test (SUT)
Input: *Nem*, *Faults*: a nemesis and the faults it can enact
Input: *Oracles*: a set of test oracles for bug detection
Input: S: number of steps in each schedule
Input: T: total time budget for testing
Output: *Bugs*: a set of bugs detected

```
1  P_f ← instrumentSystem (P_0)
2  Policy ← { initState , Faults}
3  repeat
4      curState ← initState
5      repeat
6          fault ← Policy.getNextFault ( curState )
7          Nem.enactFault(fault)
8          events ← observeSystemUnderTest (P_f)
9          timeline ← constructTimeline (events)
10         nextState ← abstractTimeline (timeline)
11         rwd ← calculateReward (curState, fault, nextState)
12         Policy ← learn (Policy, curState, fault, rwd)
13         curState ← nextState
14     until maximum steps S reached
15     resetSystemUnderTest (P_f)
16 until time budget T exhausts
17 Bugs ← Oracles.identifyBugs (events)
```

Timeline-driven Testing

Reactive Fuzzing using Q-learning



Evaluation

Research Questions

RQ.1 **Coverage achieved by Mallory:** Can Mallory cover more distinct program states than Jepsen?

RQ.2 **Efficiency of bug finding:** Can Mallory find bugs more efficiently than Jepsen?

RQ.3 **Discovering new bugs:** Can Mallory discover new bugs in rigorously-tested distributed system implementations?

Subject Programs

- Braft
- Dqlite
- MongoDB
- Redis
- ScyllaDB
- TiKV

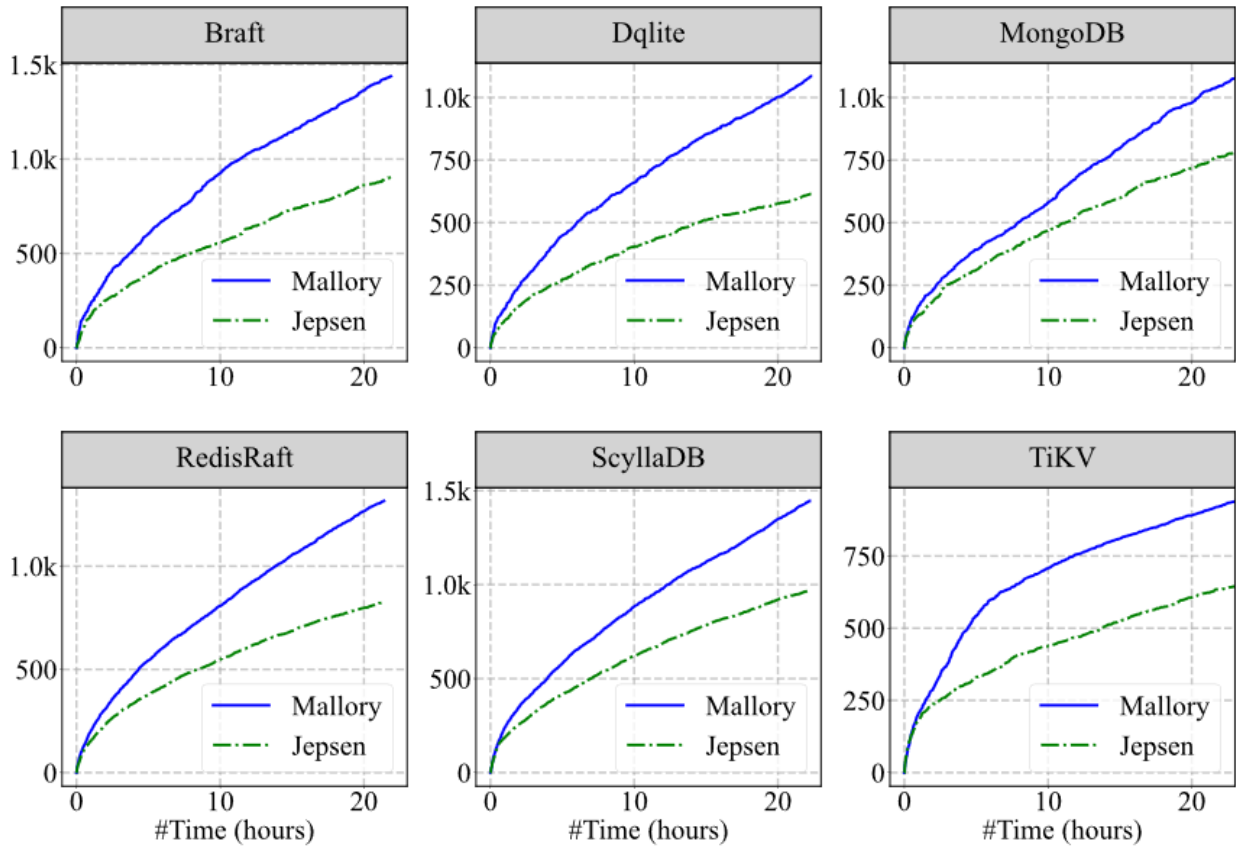
Comparison

- Jepsen

Our tool Mallory and dataset are publicly available at:

https://github.com/dsfuzz/mallory

RQ.1 State Coverage



Cover same state number 2.24× faster

Subject	State-impr	Speed-up	$\hat{A}_{12}$	U
Braft	59.34%	2.28×	1.00	<0.01
Dqlite	76.14%	2.56×	1.00	<0.01
MongoDB	36.48%	1.57×	1.00	<0.01
Redis	58.92%	2.06×	1.00	<0.01
ScyllaDB	48.82%	1.88×	1.00	<0.01
TiKV	45.93%	3.07×	1.00	<0.01
AVG	54.27%	2.24×	-	-

Cover 54.27% more states

RQ.2 Efficiency of Bug Finding

Bug ID	Type of bug	Time to exposure		
		MALLORY	JEPSEN	$\hat{A}_{12}$
Dqlite-416	Null pointer deference	**0.76h**	1.44h	**1.00**
Dqlite-356	Snapshot installing failure	T/O	T/O	0.50
Dqlite-338	Election fatal with split votes	0.16h	0.16h	0.50
Dqlite-327	Member removal failure	0.06h	0.05h	0.49
Dqlite-324	Log truncation failure	**5.94h**	T/O	**1.00**
Dqlite-323	Membership rollback failure	**8.68h**	T/O	**1.00**
Dqlite-314	Crashing on disk failure	T/O	T/O	0.50
Redis-54	Snapshot panic	**3.33h**	5.00h	**0.95**
Redis-53	Committed entry conflicting	**0.87h**	1.17h	**0.89**
Redis-51	Not handling unknown node	**1.66h**	6.40h	**1.00**
Redis-44	Loss of committed write logs	**0.34h**	0.58h	**0.60**
Redis-43	Snapshot index mismatch	0.16h	0.16h	0.50
Redis-42	Snapshot rollback failure	0.29h	0.26h	0.50
Redis-28	Split brain after node removal	**9.56h**	T/O	**1.00**
Redis-23	Aborted read with no leader	**7.29h**	T/O	**1.00**
Redis-17	Split brain and update loss	**11.06h**	T/O	**1.00**
Bugs exposed in total		14	9	-
Average time usage		6.13h	11.45h	-
Speed-up on time usage		-	1.87×	-

Find *more* bugs

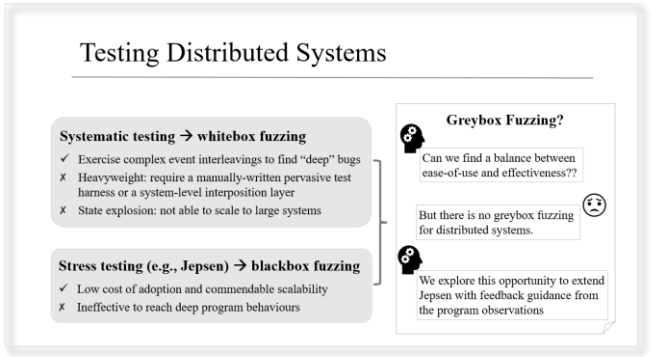
Find bugs 1.87× faster

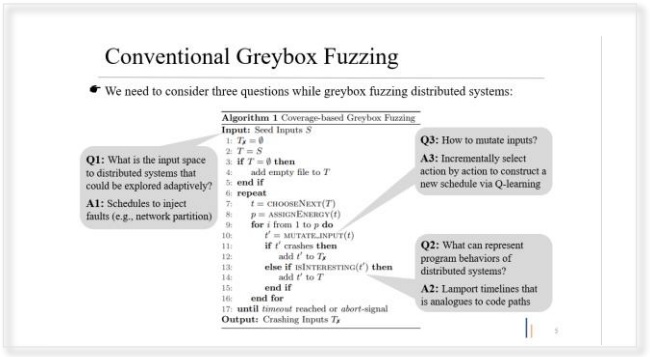
RQ.3 Discovery of New Bugs

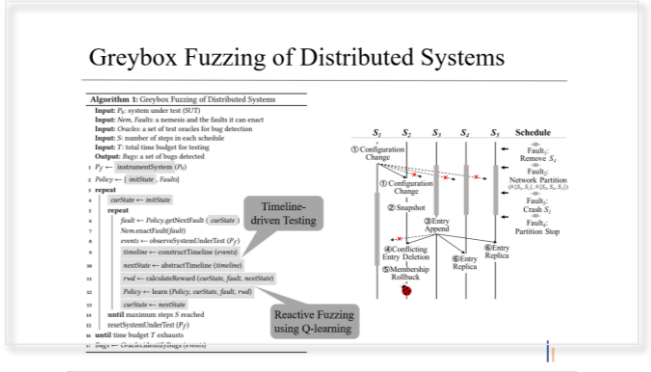
ID	Subject	Bug description	Bug checker	Bug status	JEPSEN?
1	Braft	Read stale data after a newly written update is visible to others	ELLE	Investigating	✓
2	Braft	Leak memory of the server when killed before its status becomes running	ASan	CVE-Granted, fixed	✗
3	Dqlite	Two leaders are elected at the same term due to split votes	Log checker	Confirmed	✗
4	Dqlite	No leader is elected in a healthy cluster with an even number of nodes	Log checker	Confirmed, fixed	✗
5	Dqlite	A node reads dirty data that is modified but not committed by another node	ELLE	Confirmed	✗
6	Dqlite	Lose write updates due to split brain	ELLE	Confirmed	✗
7	Dqlite	A null pointer is dereferenced due to missing the pending configuration	ASan	CVE-Requested	✓
8	Dqlite	Leak allocated memory when failing to extend entries	ASan	CVE-Requested, fixed	✗
9	Dqlite	Buffer overflow happens while restoring a snapshot	ASan	CVE-Requested	✗
10	Dqlite	A node has an extra online spare	Log checker	Confirmed	✗
11	Dqlite	Violate invariant as a segment cannot open while truncating inconsistent logs	Log checker	CVE-Requested	✗
12	MongoDB	Not repeatable read due to missing the local write update	ELLE	Confirmed	✗
13	MongoDB	Not read committed due to missing the newly written update	ELLE	Confirmed	✗
14	Redis	Read stale data after new data is written to the same key	ELLE	Confirmed	✗
15	Redis	Buffer overflow due to writing data to a wrong data structure	ASan	CVE-Granted, fixed	✗
16	Redis	Runtime panic on initializing a cluster due to database version mismatch	Log checker	CVE-Granted	✓
17	TiKV	No leader is elected for a long time in a healthy cluster	Log checker	Investigating	✗
18	TiKV	Lose write updates due to split brain	ELLE	Investigating	✗
19	TiKV	Runtime fatal error when one server cannot get context before the deadline	Log checker	CVE-Granted	✗
20	TiKV	Runtime fatal error in a server when the placement driver is killed	Log checker	CVE-Granted	✗
21	TiKV	Runtime fatal error when failing to update max timestamp for the region	Log checker	CVE-Granted	✗
22	TiKV	Monotonic time jumps back at runtime	Log checker	Investigating	✓

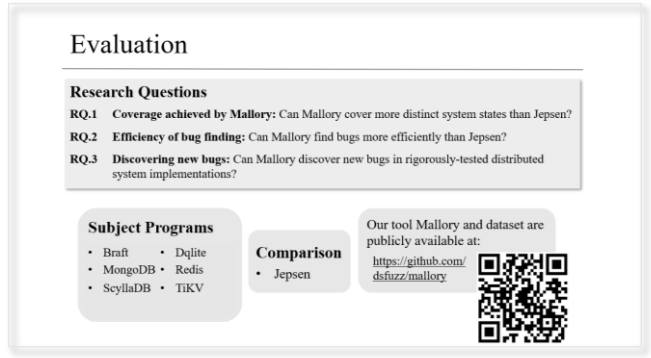
Discover 22 zero-day bugs and receive 6 CVE ID

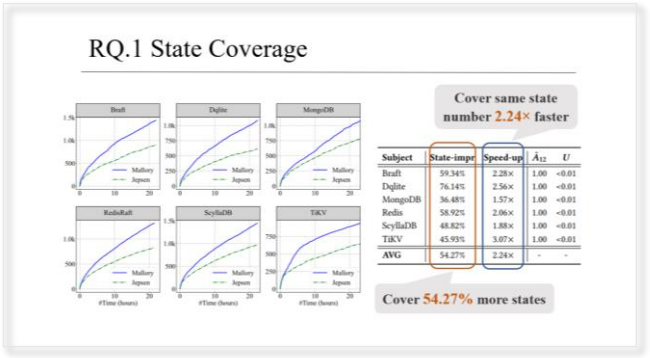
Summary



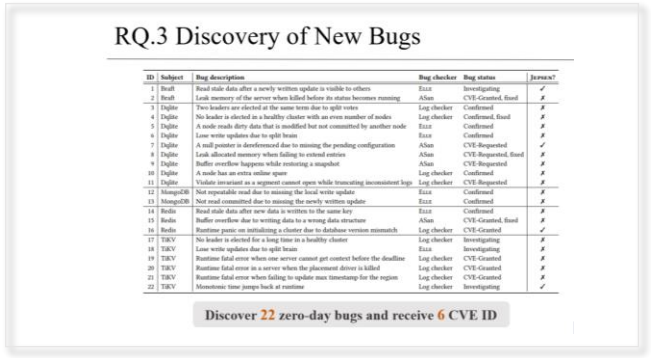








Subject	State-impr	Speed-up	$\hat{A}_{12}$	U
Braft	59.34%	2.28×	1.00	<0.01
Dqlite	76.14%	2.56×	1.00	<0.01
MongoDB	36.48%	1.57×	1.00	<0.01
Redis	58.92%	2.06×	1.00	<0.01
ScyllaDB	48.82%	1.88×	1.00	<0.01
TiKV	45.93%	3.07×	1.00	<0.01
AVG	54.27%	2.24×	-	-



THANKS!!