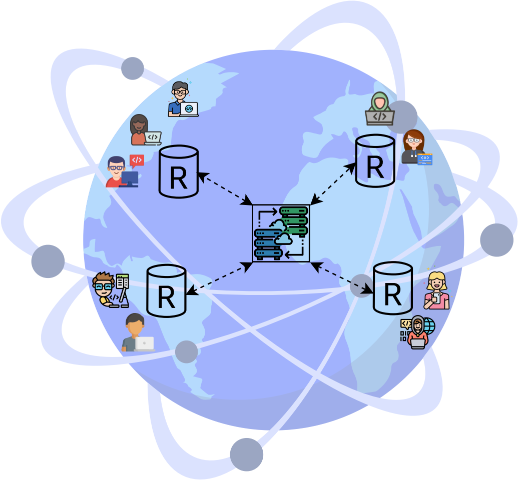
Toward Thorough and Practical Integration Testing of Replicated Data Systems

Provakar Mondal
Advisor: Dr. Eli Tilevich

Software Innovations Lab, Virginia Tech
provakar@cs.vt.edu

Table of Contents

Replicated Data System (RDS)



- High availability
- Low latency
- Partition tolerance
- Scalability

RDS in Practice

Domains

Platforms

Azure

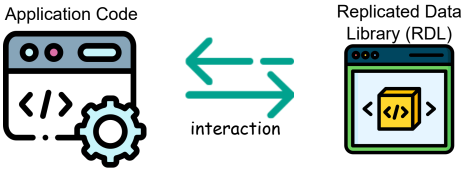
ORACLE
GOLDEN GATE

APACHE hadoop

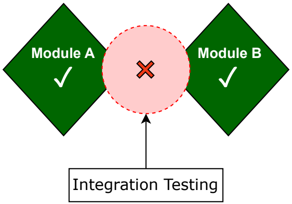
Middleware Designs

- CRDT
- MRDT
- ECRO

Integrating RDL and Testing the Integration



App ⇔ RDL Interaction



Problem Context

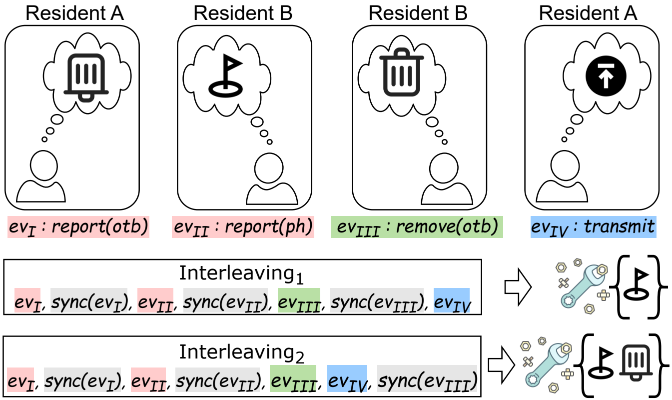
Application developers often:

- misunderstand RDL properties [1]
- hold incorrect assumptions [2]

Goal:

- ✗ Test RDL Design
- ✗ Test RDL Implementation
- ✔ Test RDL Integration

Motivating Scenario



Challenges

- Non-deterministic distributed execution
- Subtle bugs in some interleavings [3]
- Exhaustive replay for bug reproduction
- Combinatorial explosion of interleavings
- Impractical exhaustive replay

Completed Work

Exhaustive Interleavings Replay

ER-π—a middleware for exhaustive integration testing of RDL-based applications.

1. Detects distributed events raised from the app–RDL interactions.
2. Generates and persists all possible interleavings.
3. Reduces the problem space via *four* novel pruning algorithms.
4. Replays interleavings to reproduce bugs.

Pruning Algorithms

1. Event Grouping
2. Replica Specific
3. Event Independence
4. Failed Ops

Event Grouping

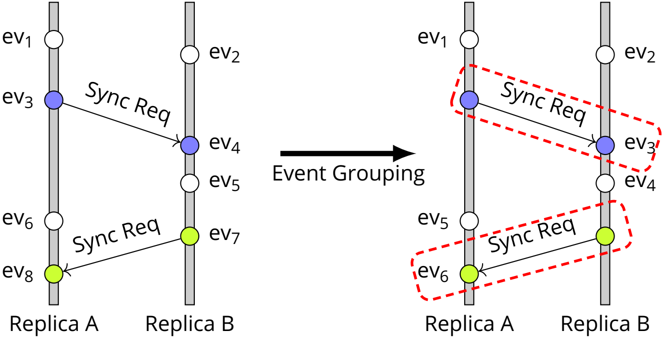


Figure: Grouping Events to Reduce Their Total #

Replica Specific

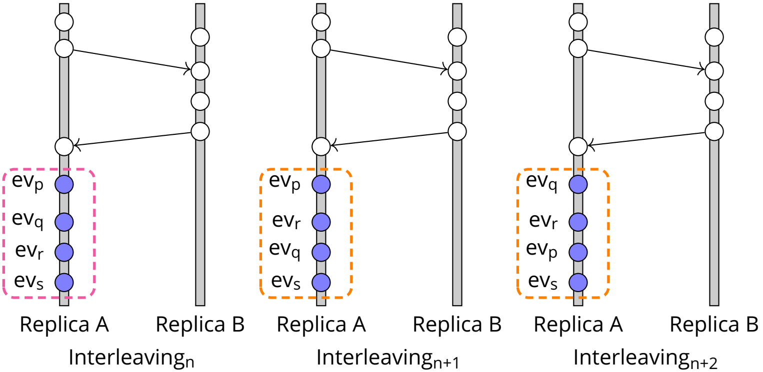
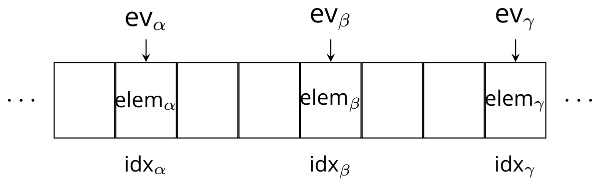


Figure: Replica B-specific Pruning

Event Independence



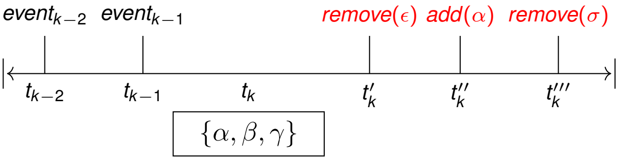
If Explored: $\{...ev_\alpha, ev_\beta, ev_\gamma...\}$

Consider Explored: $\{...ev_\alpha, ev_\gamma, ev_\beta...\}$

Consider Explored: $\{...ev_\beta, ev_\alpha, ev_\gamma...\}$

Figure: Event Independence Pruning

Failed Ops



If Explored: $\{...remove(\epsilon), add(\alpha), remove(\sigma)...\}$

Consider Explored: $\{...remove(\epsilon), remove(\sigma), add(\alpha)...\}$

Consider Explored: $\{...add(\alpha), remove(\epsilon), remove(\sigma)...\}$

Figure: Failed Ops Pruning

Motivating Example Revisited

1. $ev_I : report(otb)$
2. $sync(ev_I)$
3. $ev_{II} : report(ph)$
4. $sync(ev_{II})$
5. $ev_{III} : remove(otb)$
6. $sync(ev_{III})$
7. $ev_{IV} : transmit$

---

7 Events $\equiv$ 5040 Interleavings

- Sync event causally depends on the corresponding update event
  - i) $(ev_I, sync(ev_I)) \Rightarrow ev'_I$
  - ii) $(ev_{II}, sync(ev_{II})) \Rightarrow ev'_{II}$
  - iii) $(ev_{III}, sync(ev_{III})) \Rightarrow ev'_{III}$
  - iii) $ev_{IV} : transmit$

---

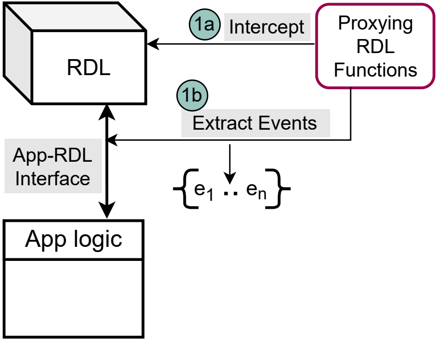
4 events $\equiv$ 24 Interleavings

- $\{ev_{IV}, \underbrace{\{ev'_I, ev'_{II}, ev'_{III}\}}_{3!=6}\} \rightarrow$ Problems: $\{\} \Rightarrow$ 1 Interleaving

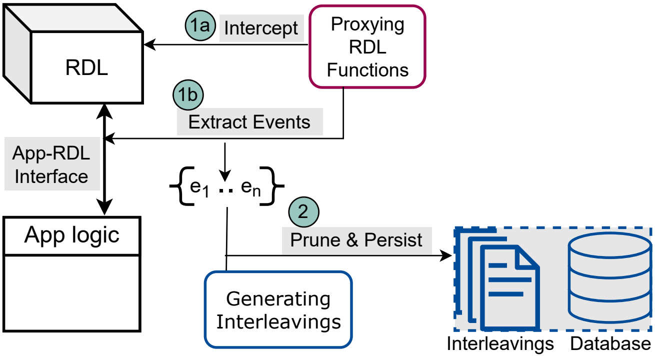
Total Interleavings: $24 - 5 = 19$

Problem Space Reduction: $\lfloor \frac{5040}{19} \rfloor = \mathbf{265}\times$

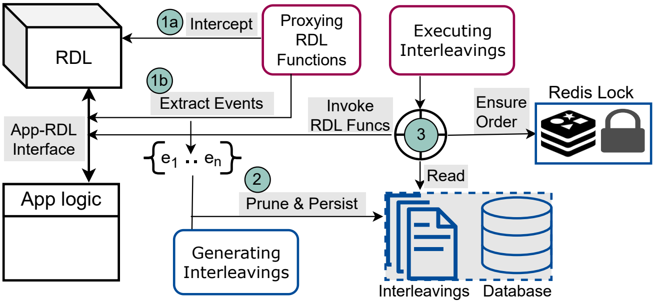
ER-π: System Components and Workflow



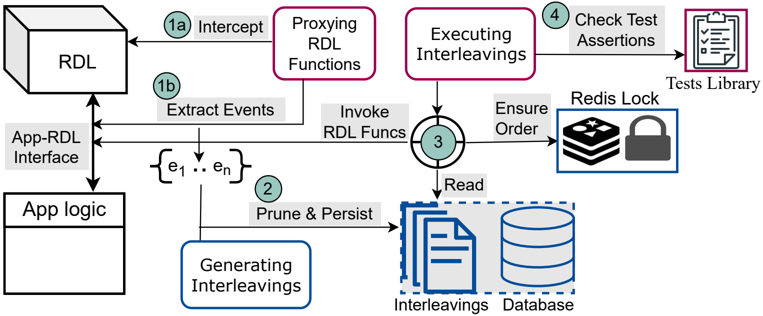
ER-π: System Components and Workflow



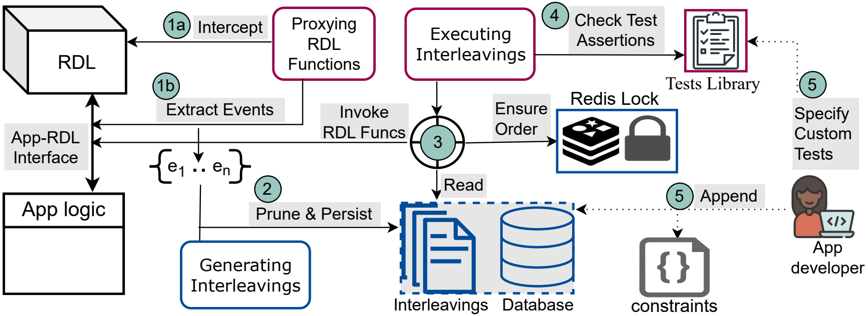
ER-π: System Components and Workflow



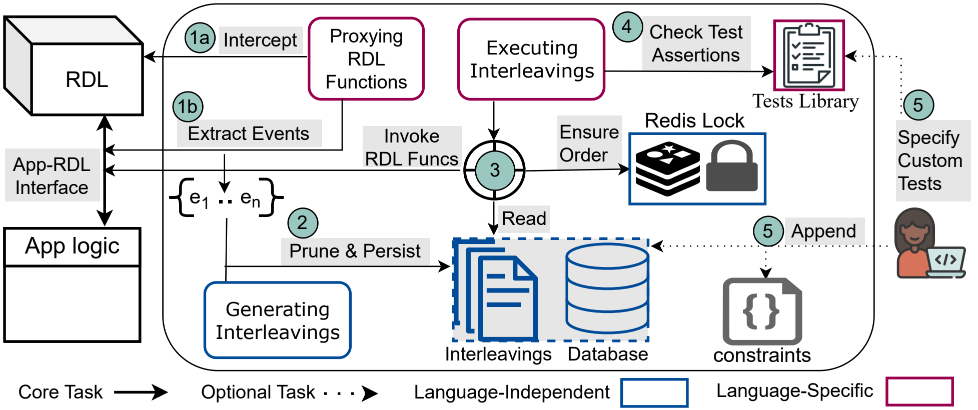
ER-π: System Components and Workflow



ER-π: System Components and Workflow



ER-π: System Components and Workflow



Implementation

Language-

- Specific Components: Go(≈ 300), JavaScript(≈ 280), and Java(≈ 415)
- Agnostic Components: C++($\approx 2K$), Datalog ($\propto$ # of interleavings)

Distributed Lock: Redis Lock [4]

Research Questions

- ✓ RQ1: Reproducing Bugs
- ✓ RQ2: Recognizing Misconceptions
- ✓ RQ3: Reducing Problem Space

RQ1: Reproducing Bugs

BugName	Issue#	#Events	Status	Reason
1. Roshi-1	18	9	closed	misconception
2. Roshi-2	11	10	closed	RDL issue
3. Roshi-3	40	21	closed	misconception
4. OrbitDB-1	513	12	open	—
5. OrbitDB-2	512	8	open	—
6. OrbitDB-3	1153	15	closed	misuse
7. OrbitDB-4	583	18	closed	misconception
8. OrbitDB-5	557	24	closed	misconception
9. ReplicaDB-1	79	10	closed	misuse
10. ReplicaDB-2	23	14	closed	misconception
11. Yorkie-1	676	17	open	—
12. Yorkie-2	663	22	closed	misconception

Table: Bug benchmarks. ``#Events”—# of interleaved events. ``Status''—if the bug is closed by library developers. ``Reason''—what causes the bug.

RQ2: Recognizing Misconceptions

1. The underlying network ensures causal delivery [5].
2. The order of List elements is always consistent [5].
3. Moving items in a List doesn't cause duplication [6].
4. Sequential IDs are always suitable for creating new items in a to-do list [6].
5. Multiple replicas in different regions mathematically resolve to the same state without coordination [7, 2].

RQ3: Reducing Problem Space

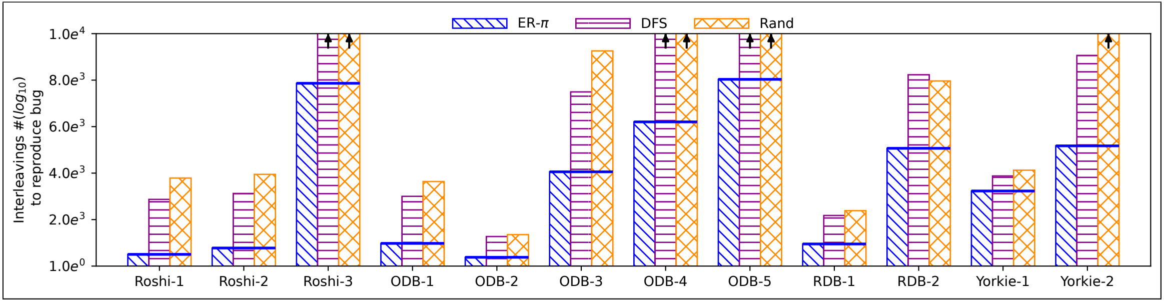


Figure: Number of Interleavings Explored to Reproduce Bug

RQ3: Reducing Problem Space

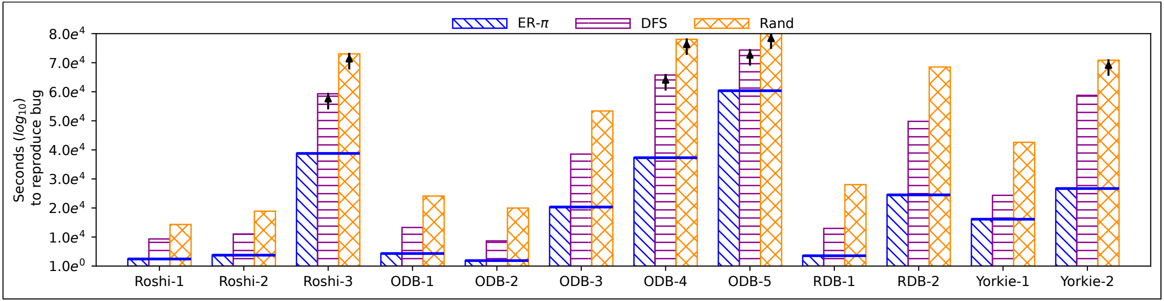


Figure: Time Required to Reproduce Bug

Limitations

- SUT comprises distinct RDL and business logic components
- Pruning algorithms assume eventual consistent RDLs
- Interfacing with RDLs relies on robust proxying support

Insights and Implications

Key Insights

- Non-deterministic interleavings can cause subtle bugs
- Misconceptions about App-RDL interactions are common
- Exhaustive interleaving replay is feasible with effective pruning
- Middleware for integration testing can benefit RDL applications

Implications

- Middleware for replaying interleavings is useful, even without pruning
- Testing RDL-based applications requires tools like ER-π
- Future RDL design should strive to minimize usage misconceptions

Ongoing Work

Prioritizing Interleavings Replay

Motivation: Even with `pruning`, the remaining interleavings may still be:

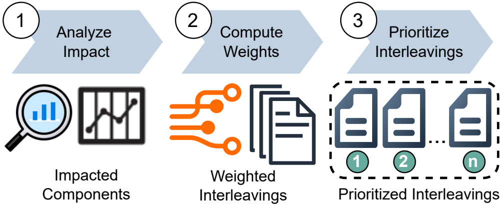
- ★ prohibitively large
- ★ expensive to replay exhaustively
- ★ ineffective for exposing bugs due to the latest code changes

Goal: Prioritize the replay of high-impact interleavings

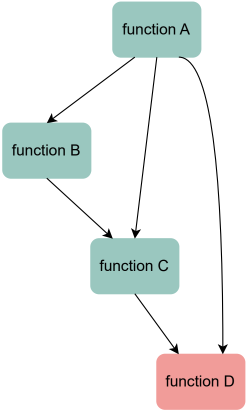
Prioritizing Approach

Order $\mathcal{I}$ into $\mathcal{I}'$ such that:
$\mathcal{I}' = \langle i'_1, \ldots, i'_n \rangle \Rightarrow \text{weight}(i'_1) \geq \ldots \geq \text{weight}(i'_n)$

1. Impact analysis
2. Weight computation
3. Prioritization and replay



Step 1: Impact Analysis



- Construct the system's call graph
- Find impacted components via Change Impact Analysis (CIA) [8]
- Identify the events generated by each affected component
- Treat the identified events with greater caution

Step 2: Weight Computation

Goal: Assign each interleaving a weight indicating its likelihood to trigger a bug.

What determines weight?

- Does the interleaving involve impacted events?
- If so, how many?
- Are they co-located?

Solution outline:

- To compute weights effectively, we need a suitable programming model
- Probabilistic logic Programming (ProbLog) fits well due to:
  - Declarative nature
  - Probabilistic reasoning support
  - Suitable abstractions to encode dependencies between impacted events

Step 3: Prioritization

- Sort interleavings based on weights
- Replay highest-weighted first
- Enable guided system replay

Toward a Unified Solution

Unified Vision

1. ER-π $\rightarrow$ Thorough integration testing by *providing infrastructure* and *pruning*
2. Ongoing work $\rightarrow$ Practical integration testing by *prioritizing*

Flexible Use: Each technique can be used

- Individually à la carte
- In combination
- In any order, depending on testing needs.

References I

[1] Yicheng Zhang, Matthew Weidner, and Heather Miller. "Programmer Experience When Using CRDTs to Build Collaborative Webapps: Initial Insights". In: *13th annual workshop on the intersection of HCI and PL (PLATEAU)*. Mar. 2023. DOI: 10.1184/R1/22277341.v1.

[2] Shadaj Laddad et al. "Keep CALM and CRDT On". In: *Proceedings of the VLDB Endowment* 16.4 (2022), pp. 856–863.

[3] Ohad Shacham, Mooly Sagiv, and Assaf Schuster. "Scaling Model Checking of Dataraces Using Dynamic Information". In: *Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming*. 2005, pp. 107–118.

[4] Redis Documentation. *Distributed Locks with Redis*. https://redis.io/docs/latest/develop/use/patterns/distributed-locks/. 2023.

[5] Yuqi Zhang et al. "MET: Model Checking-Driven Explorative Testing of CRDT Designs and Implementations". In: *arXiv preprint arXiv:2204.14129* (2022).

References II

[6] Andrew Jeffery and Richard Mortier. "AMC: Towards Trustworthy and Explorable CRDT Applications with the Automerge Model Checker". In: *Proceedings of the 10th Workshop on Principles and Practice of Consistency for Distributed Data*. 2023, pp. 44–50.

[7] Leena Joshi. *How to simplify distributed app development with CRDTs*. Accessed 2024. 2022.

[8] Xiaobing Sun et al. "Change impact analysis based on a taxonomy of change types". In: *2010 IEEE 34th Annual computer software and applications conference*. IEEE. 2010, pp. 373–382.