Software Testing as a Sampling Problem

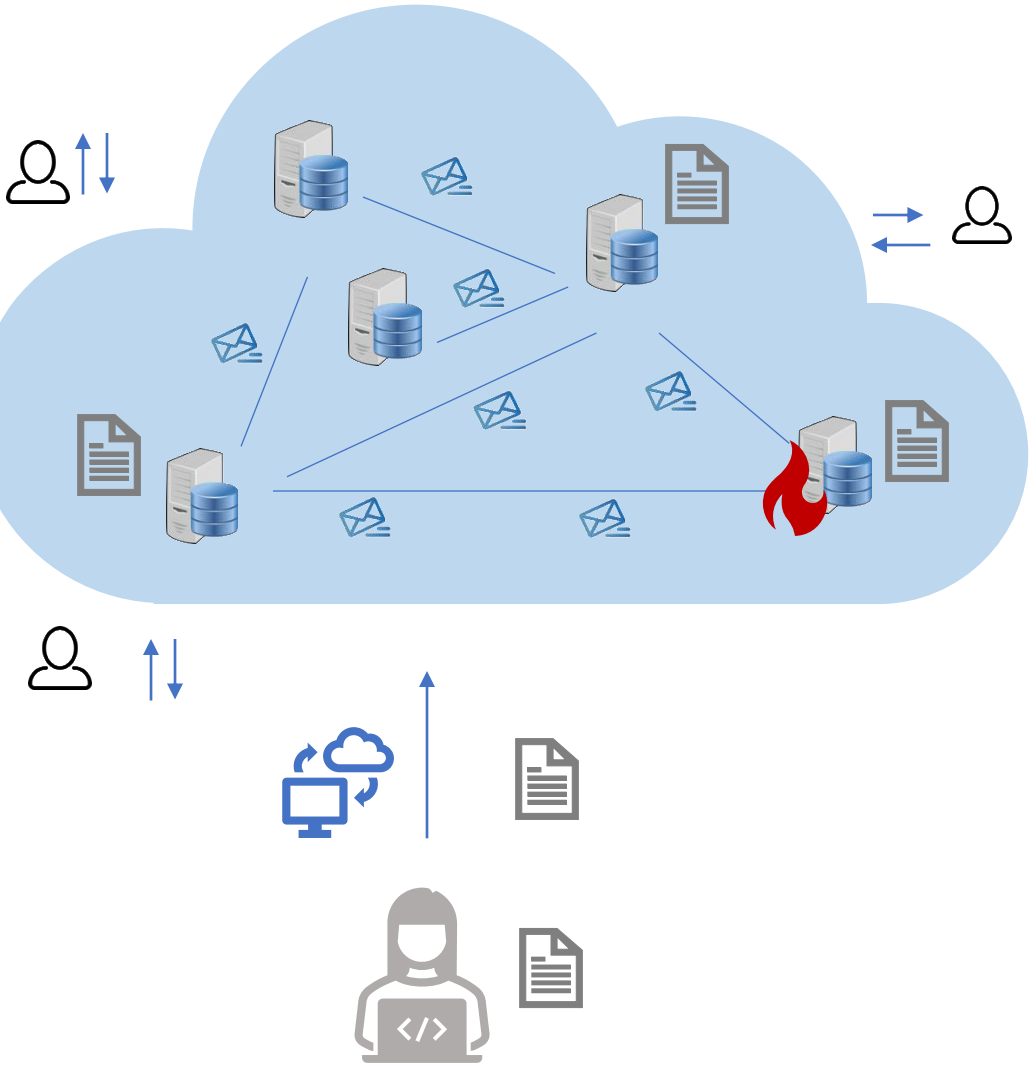
Burcu Kulahcioglu Ozkan

b.ozkan@tudelft.nl

https://burcuku.github.io/home/

FORSE TU Delft

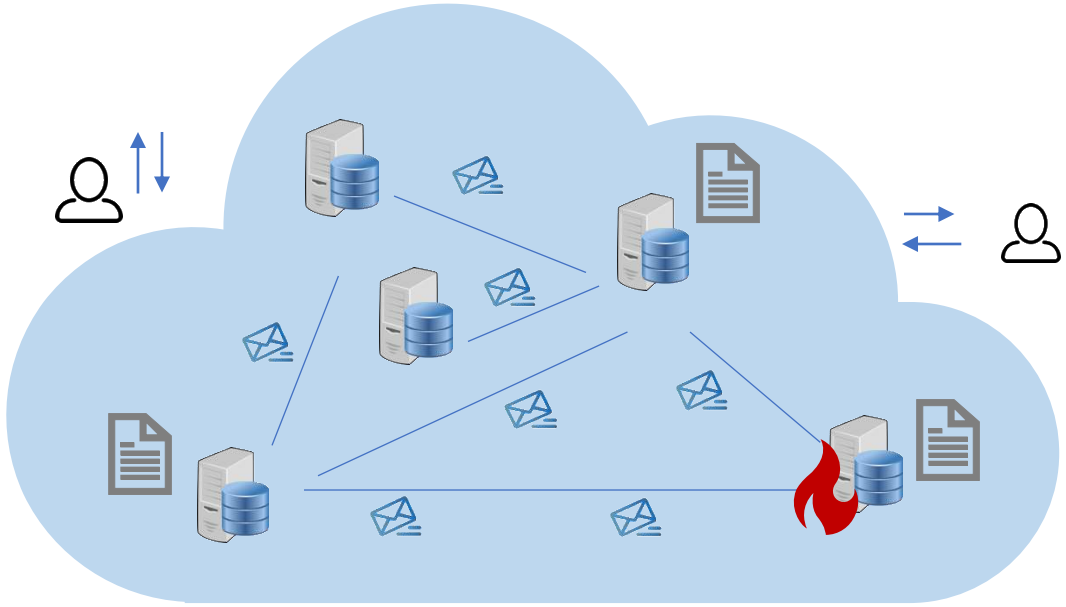
Modern software systems are inherently complex



Correctness depends on:

- Sequential correctness
- Concurrency correctness
- Fault-tolerance/resilience

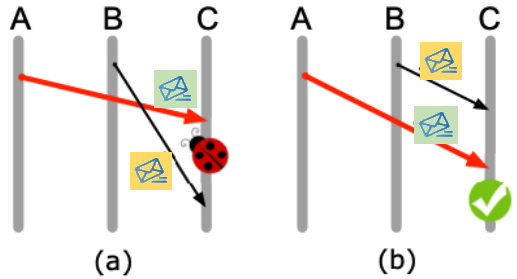
Modern software systems are inherently complex highly concurrent and distributed



Need to reason about:

- Concurrency
- Asynchrony
- Network faults
- Process faults

Unexpected orderings, buggy executions:



Hadoop Map/Reduce / MAPREDUCE-3274

Race condition in MR App Master Preemtion can cause a dead lock

https://issues.apache.org/jira/browse/MAPREDUCE-3274

"That is one monster of a race!" — m3274

A taxonomy of nondeterministic concurrency bugs [Leesatapornwongsa et. Al., ASPLOS'16]

Many bugs due to concurrency nondeterminism

Cassandra / CASSANDRA-9794

Linearizable consistency for lightweight transactions is not achieved

ZooKeeper / ZOOKEEPER-4003

Zookeeper server breakdown Frequently

Kafka / KAFKA-382

Write ordering guarantee violated

HBase / HBASE-2849

HBase clients cannot recover

ActiveMQ / AMQ-2780

ActiveMQ not preserving Message Order

Core Server / SERVER-37948

Linearizable read concern is not satisfied by getMores on a cursor

Hadoop Map/Reduce / MAPREDUCE-3274

Race condition in MR App Master Preemtion can cause a dead lock

Core Server / SERVER-38084

MongoDB hangs when a part of a replica set

ActiveMQ / AMQ-6911

Constraint violation on failover (Postgresql)

Fuzzing redisraft crash and assertion failure #648

Open ds-testing-user opened this issue on Oct 19, 2023 · 0 comments

Challenges of testing distributed systems

(C0) Test oracle
- What is the correctness specification?

→ Assertion violations
Generic properties (e.g., serializability)

(C1) Test harness discovery
- What are the requests/transactions to submit?

→ A few random client transactions

(C2) Enumerating executions
- What interleavings of events to exercise?

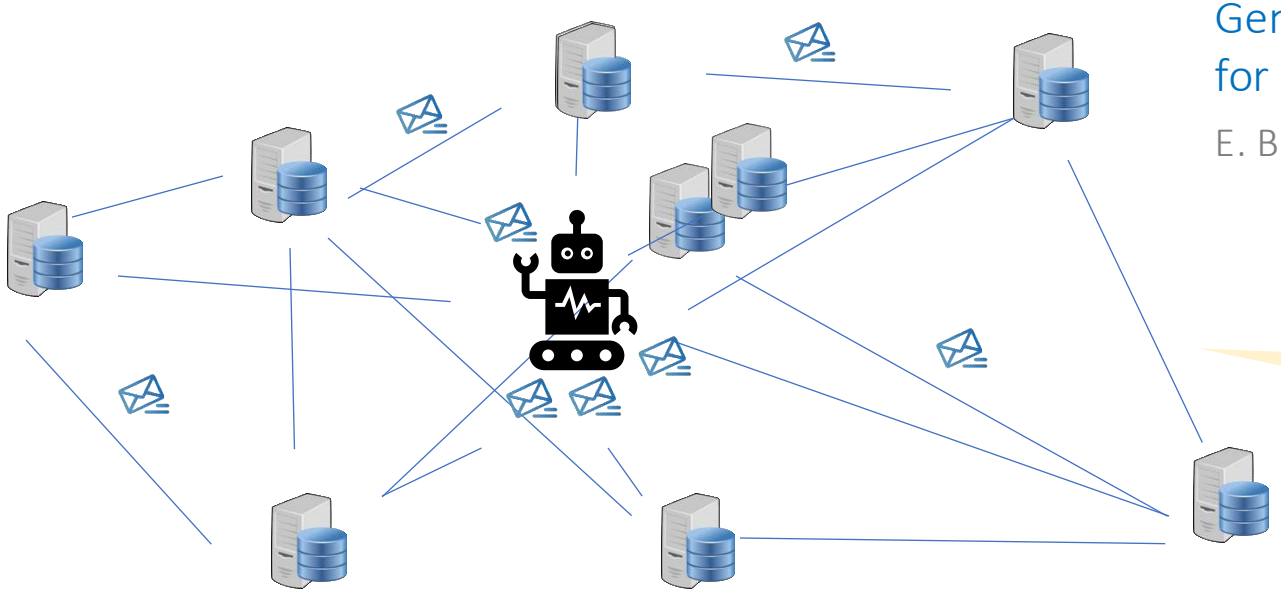
→ How to explore possible executions efficiently?
Combinatorial complexity!

(C3) Improving interpretability
- Is the buggy trace easy to understand?

→ How to produce understandable traces?

Controlled concurrency testing + fault injection

- Control the non-determinism in a test environment
- Design testing strategies to explore different program executions
- Reproduce a buggy execution for easier debugging



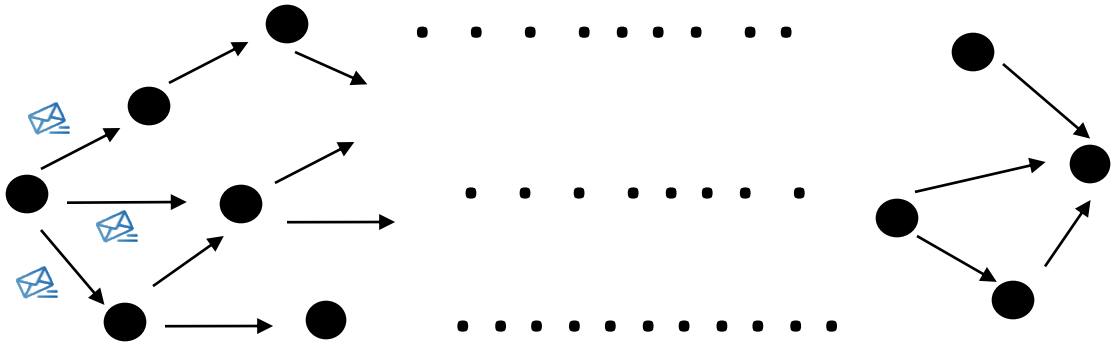
Generalized Concurrency Testing Tool for Distributed Systems

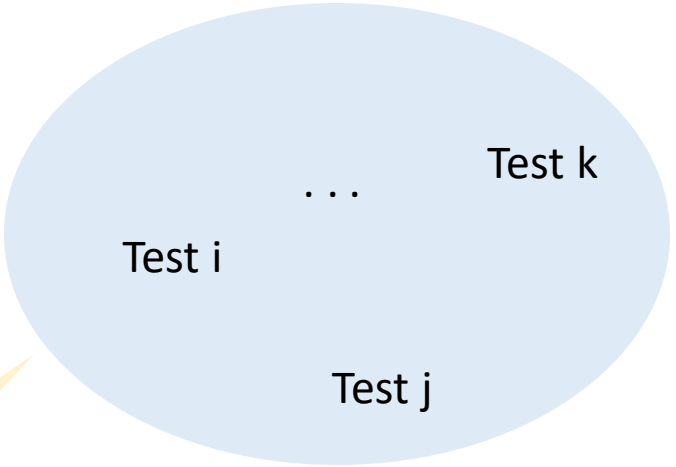
E. B. Gulcan, J. Neto, B. Kulahcioglu Ozkan [ISSTA'24]

(C2) What event orderings to test?

Concurrency testing as a sampling problem

An enormous number of possible event orderings





The set of all possible executions

Infeasible to test all

Testing is sampling from that set

How to characterize this set of event orderings?

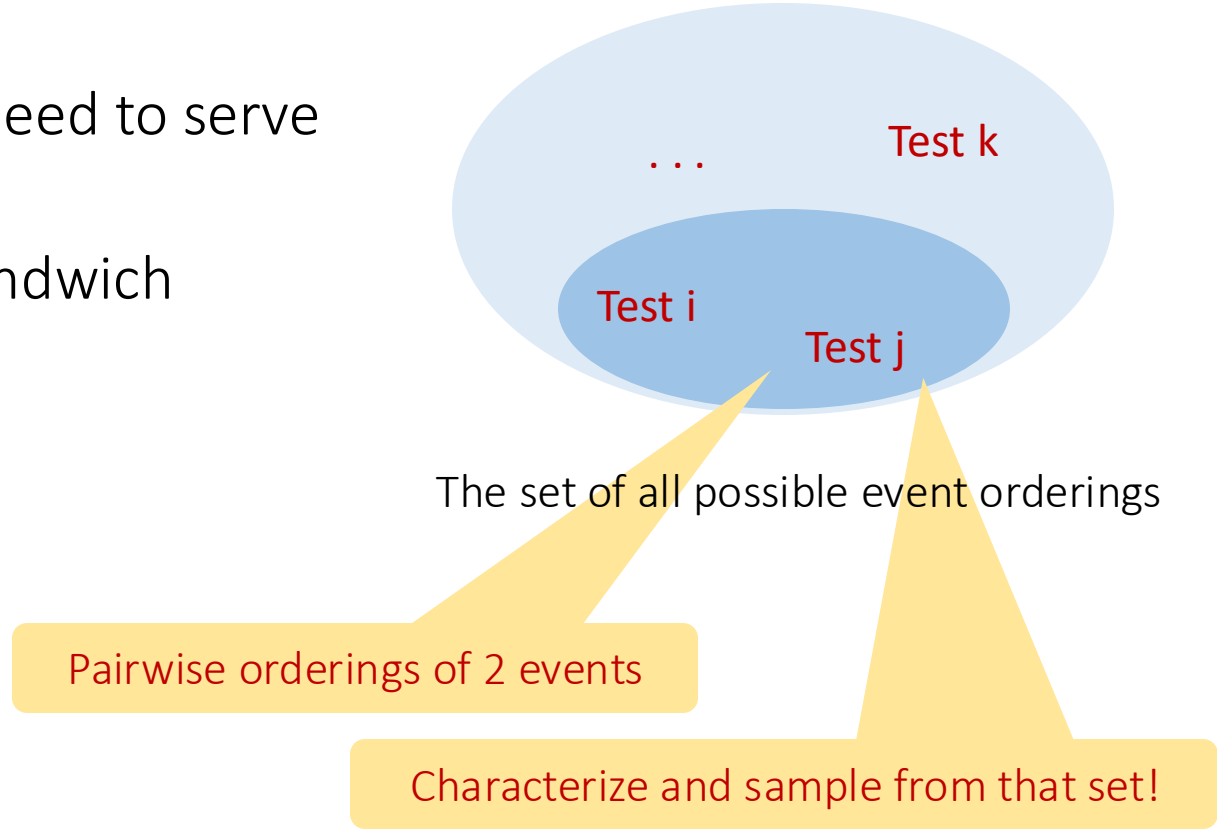
A glimpse of characterizing the sample set with nontrivial guarantees

Serving sandwiches to a group of **n** people

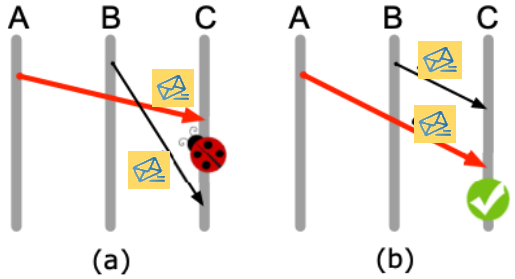
How many rounds of sandwiches do we need to serve to ensure that:

For each pair of people, one gets their sandwich before the other in at least one round?

❓ n! ❓ < n! ❓ only 2



Revisiting the example concurrency bug:

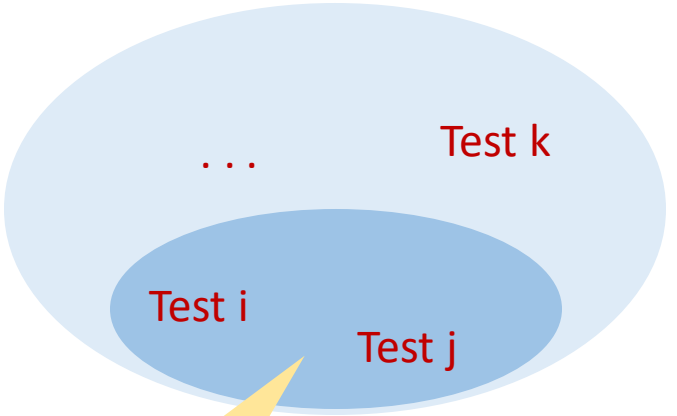


Hadoop Map/Reduce / MAPREDUCE-3274

Race condition in MR App Master Preemtion can cause a dead lock

"That is one monster of a race!" — m3274

https://issues.apache.org/jira/browse/MAPREDUCE-3274



The set of all possible event orderings

Pairwise orderings of 2 events

A glimpse of characterizing the sample set (cont.)

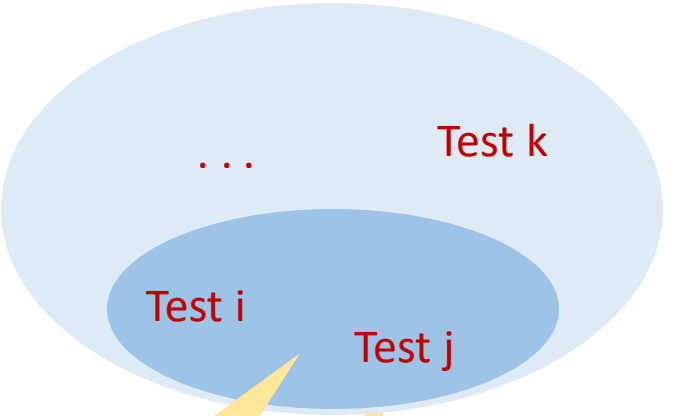
Let's make it more interesting:

How many rounds of sandwiches do we need to serve to ensure that:

For each pair of people, one gets their sandwich before the other in at least one round such that:

everyone gets their sandwich before their promoters?

❓ n! ❓ < n! ❓ only 2



The set of all possible event orderings

Pairwise orderings of 2 events

Characterize and sample from that set!

Our notion of concurrency bug depth:

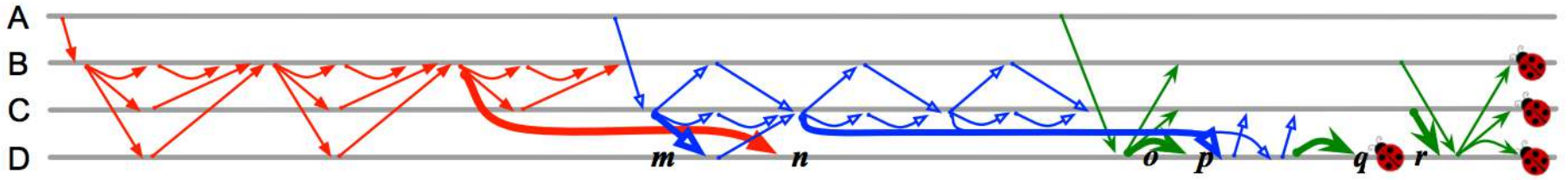
Number of minimum ordering requirements between events

- $\langle e_1, e_2 \rangle$ e.g. order violation

- $\langle e_1, e_2, e_3 \rangle$ e.g. atomicity violation

…

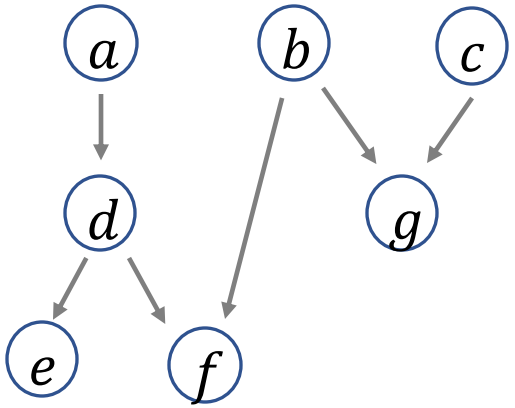
- $\langle e_1, \ldots, e_n \rangle$ more complicated bugs



Bug in Cassandra 2.0.0 *(img. from Leesatapornwongsa et. al. ASPLOS'16)*

Challenging to characterize event orderings of distributed systems:

- Upgrowing, causally ordered set of events, revealed during execution
- Mutual dependency to the schedule



Schedule: $a\,d\,e\,b\,f\,c\,g$

- Build a schedule online
- For an arbitrary ordering

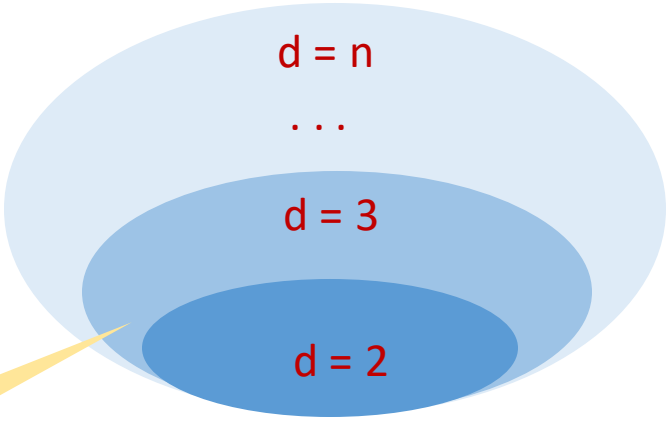
Use combinatorial results for posets!

Follow up works using ideas from model checking and verification

Concurrency testing as sampling (with nontrivial probabilistic guarantees)

Online sampling algorithm from a set of test cases that:

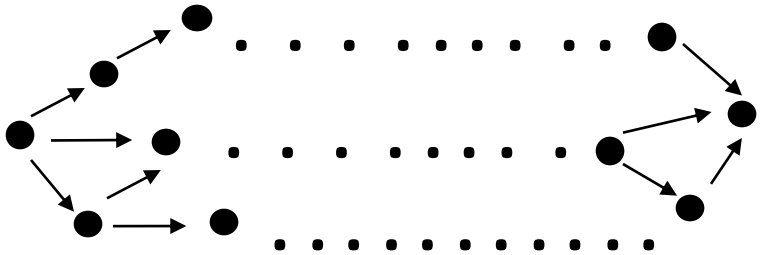
- Order **d** of **n** number of events in a particular way
- Arbitrary (causal) dependencies between distributed system events
- Events/execution space revealing online



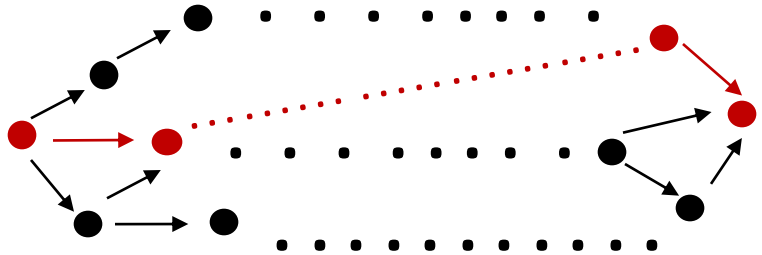
The set of all possible event orderings

Characterize and sample from that set!

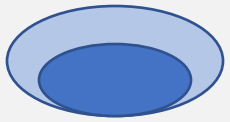
Some highlights from sampling based testing



Systematic testing – infeasible

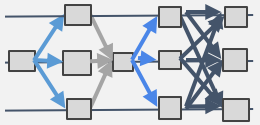


Randomized testing with theoretical guarantees



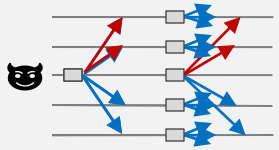
Randomized testing of distributed systems with theoretical guarantees

B. Kulahcioglu Ozkan, R. Majumdar, F. Niksic, M. T. Befrouei, G. Weissenbacher [OOPSLA'18, Distinguished paper]



Transferring ideas from verification of **consensus systems**

C. Dragoi, C. Enea, B. Kulahcioglu Ozkan, R. Majumdar, F. Niksic [OOPSLA'20]

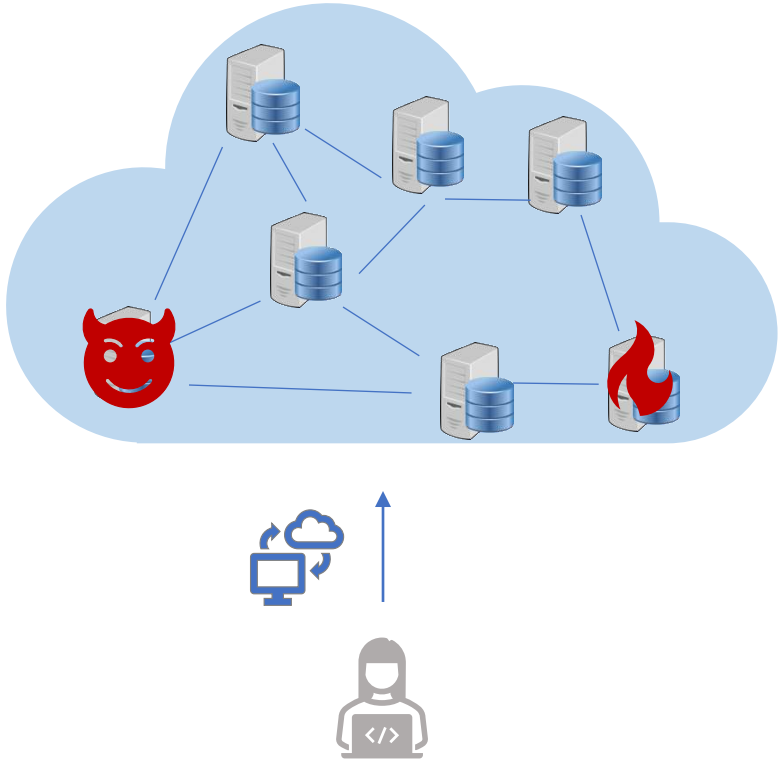


Incorporating Byzantine faults for testing **blockchain systems**

L. N. Winter, F. Buse, D. de Graaf, K. von Gleissenthall, B. Kulahcioglu Ozkan [OOPSLA'23, Distinguished paper]

Bugs in the (earlier versions of) blockchain algorithms

Unexpected adversary scenarios violate correctness



Security Analysis of Ripple Consensus

Ignacio Amores-Sesar
University of Bern, Switzerland
ignacio.amores@inf.unibe.ch

Christian Cachin
University of Bern, Switzerland
christian.cachin@inf.unibe.ch

Jovana Mićić
University of Bern, Switzerland
jovana.micic@inf.unibe.ch

Force-Locking Attack on Sync Hotstuff

Atsuki Momose
Nagoya University

Jason Paul Cruz
Osaka University

Making Reads in BFT State Machine Replication Fast, Linearizable, and Live

Christian Berger
Universität Passau
Passau, Germany

Hans P. Reiser
Universität Passau
Passau, Germany

Alysson Bessani
LASIGE, Faculdade de Ciências, Universidade de Lisboa, Portugal

Is Stellar As Secure As You Think?

Minjeong Kim
KAIST
mjkim9334@kaist.ac.kr

Yujin Kwon
KAIST
dbwls8724@kaist.ac.kr

Yongdae Kim
KAIST
yongdaek@kaist.ac.kr

Revisiting Fast Practical Byzantine Fault Tolerance

Ittai Abraham, Guy Gueta, Dahlia Malkhi
VMware Research

with:
Lorenzo Alvisi (Cornell),
Rama Kotla (Amazon),
Jean-Philippe Martin (Verily)

Dissecting Tendermint

Yackolley Amoussou-Guenou[1,2], Antonella Del Pozzo[1], Maria Potop-Butucaru[2], and Sara Tucci-Piergiovanni[1]

[1] CEA LIST, PC 174, Gif-sur-Yvette, 91191, France
[2] Sorbonne Université, CNRS, LIP6, F-75005 Paris, France

Refactor getTrustedForLedger() #4424

Merged intelliot merged 1 commit into `XRPLF:develop` from `levinwinter:refactor-get-trusted-for-ledger` on Feb 23

Conversation 19 · Commits 1 · Checks 4 · Files changed 5

levinwinter commented on Feb 16 • edited — Contributor

This pull request fixes a bug that was detected automatically by the ByzzFuzz testing algorithm, as described in our accompanying paper Randomized Testing of Byzantine Fault Tolerant Algorithms. Florena Buse (@florenabuse), Burcu Kulahcioglu Ozkan (@burcuku), and Levin N. Winter (@levinwinter) directly contributed to this work.

High Level Overview of Change

The `getTrustedForLedger()` method has been refactored to look for validations associated with a specific ledger ID and sequence number.

Context of Change

From now on, the method can be used to filter for specific sequence numbers which was adapted throughout the codebase.

Type of Change

- [x] Refactor (non-breaking change that only restructures code)

`Refactor getTrustedForLedger()` ··· Verified ✓ a3d9be7

seelabs approved these changes on Feb 17 — View reviewed changes

seelabs left a comment — Collaborator

👍 Code is clean and straight-forward. Thanks for the patch!

❤️ 1

mtrippled suggested changes on Feb 18

- `src/ripple/app/consensus/RCLConsensus.cpp` — Show resolved
- `src/ripple/app/ledger/impl/LedgerMaster.cpp` — Show resolved
- `src/ripple/app/misc/NegativeUNLVote.cpp` — Show resolved

intelliot requested a review from **nbougalis** 8 months ago

mtrippled approved these changes on Feb 22 — View reviewed changes

mtrippled left a comment — Collaborator

👍 Looks good to me! The testing lo

❤️ 1

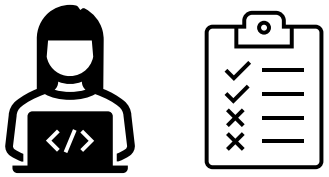
Awarded by Ripple's Bug Bounty

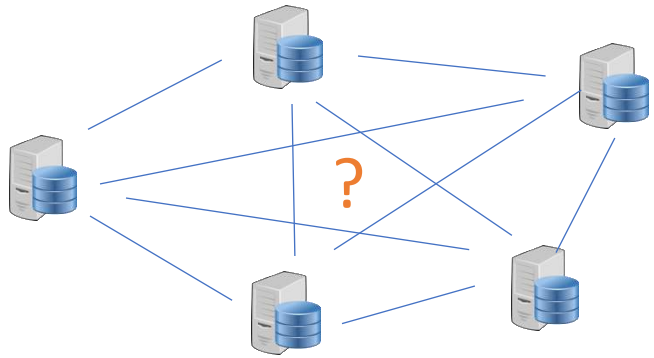
https://github.com/XRPLF/rippled/pull/4424

from "Research Highlights at UBRI Connect'25, Singapore

Next question: How to assess and increase coverage of explolation?

- ✓ Practical, automated testing
- ✓ Efficient at finding bugs



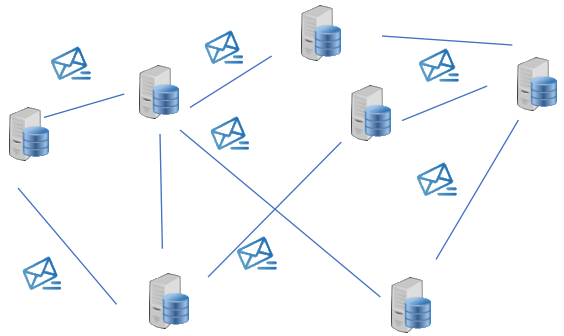


- ? How thoroughly is my system tested?
- ? What behaviors are exercised?

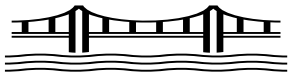
- ➢ Proposal: Coverage goal and coverage-guided exploration of distributed systems

amazon research awards

Use abstract formal models to define coverage goals



Generate tests on the implementation



Guided by the formal model used for verification

Towards bridging the gap:
implemented software ~ verified model

Model-guided fuzzing of distributed systems

E. B. Gulcan, B. Kulahcioglu Ozkan,
R. Majumdar, S. Nagendra [OOPSLA'25]

Recap: Testing as a sampling problem

Thank you!
☺

Research: How can we design testing algorithms that

- Are are applicable to large-scale systems?
- Provide theoretical guarantees on finding bugs

Approach: Testing as a sampling problem:

- Characterization and parametrization of test space
- Automated sampling algorithm to sample from the set