Distributed Test Case Generation using Model Inference with Dara

**Model Inference**

- Traces
- System Model
- Model Refinement
- Promella Model
- To SPIN

**Transparent Model Checking Design**

- Instrumented Program
- Instrumented Program
- Modified Go Runtime
- Modified Go Runtime
- Communication Layer
  - OS (Linux)
  - Global Scheduler
    - Failure Simulation
    - Virtual Clock
    - Global Assertions
    - GoRoutine State
    - Abstract Schedule

- Related Work
  - Testing: Unit, Integration, Stress
  - Modeling Languages: TLA+[TOPLAS'94], SPIN [ECBS'05], COQ [INRIA'04]
  - Verification: IronFleet [SOSP'15], Verdi [PLDI'15], Chapar [POPL'16]
  - Transparent Model Checkers: MODIST [NSDI'09],[SOSP'11], CHESS [OSDI'08]

**Developer Difficulties**

- Checking the correctness of distributed systems is HARD!
- Massive state space: low testing coverage
- Non-Deterministic Bugs: inexplicable crashes
- Partial Failures: complex corner cases

**Related Work**

- Testing: Unit, Integration, Stress
- Modeling Languages: TLA+[TOPLAS'94], SPIN [ECBS'05], COQ [INRIA'04]
- Verification: IronFleet [SOSP'15], Verdi [PLDI'15], Chapar [POPL'16]
- Transparent Model Checkers: MODIST [NSDI'09],[SOSP'11], CHESS [OSDI'08]

**Two approaches to model checking systems**

1) Abstract: Check abstract system model e.g. SPIN
   - Fast & Deep state expiration, implementation can admit bugs.
2) Concrete: Check system implementation directly e.g. MODIST
   - Sound, but Shallow state space exploration, some deep bugs go undetected.

**Dara**

- Combine both approaches, infer model from program traces, check abstract bugs with SPIN.
- Validate abstract bugs by replaying trace with transparent model checker. Iteratively refine abstract model using new traces discovered during replay.

**Dara Model Checking Workflow**

1. Traverse program AST’s and inject logging to capture messages, and specified variables
2. Run tests to generate a bootstrapping set of logs for the model inference engine
3. Compose state traces into a finite state machine (FSM). Inject invariants into FSM.
4. Model check FSM with SPIN. Convert fail-stop, deadlock, and invariant-violating traces to schedules.
5. Replay schedules, and control IO. Report replicated bugs. Send non-failing traces back to model inference.