Developing Correctly Replicated Databases Using Formal Tools

Nicolas Schiper, **Vincent Rahli**, Robbert Van Renesse,
Mark Bickford, and Robert L. Constable

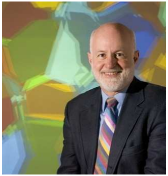
May 30, 2017

PRL & System Groups

PRL group

Mark Bickford

Robert L. Constable



Richard Eaton

Vincent Rahli

System group

Robbert van Renesse

Nicolas Schiper

Goals

What we strive for:

A platform to develop provably correct programs.

Our current interest:

Specify, verify, and generate distributed systems using formal tools. (As part of the CRASH project funded by DARPA.)

➲ Today applications are distributed over many machines.

➲ Even critical applications used by governments, banks, armies, etc.

Goals

Correctness?

How can we make sure that these applications are correct?

Distributed programs are **hard to specify, implement, and reason about**.

➲ We need to tolerate failures.

➲ It is hard to test all possible scenarios.

➲ State space explosion using model checking.

➲ Model checking often done on abstractions of the code rather than on the code itself.

We use a proof assistant (Nuprl) that implements a constructive type theory.

Achievements

- A logic of events implemented in Nuprl.

- Specified, verified, and generated **consensus protocols** (e.g., Paxos).

- **Aneris**: a total ordered broadcast service [RSR$^+$12].

- **ShadowDB**: a replicated database with 2 parametrizable replication protocols (PBR & SMR) built on top of Aneris [SRR$^+$12].

- Improved performance without introducing bugs [RBA13].

- We get **decent performance**.

Table of contents

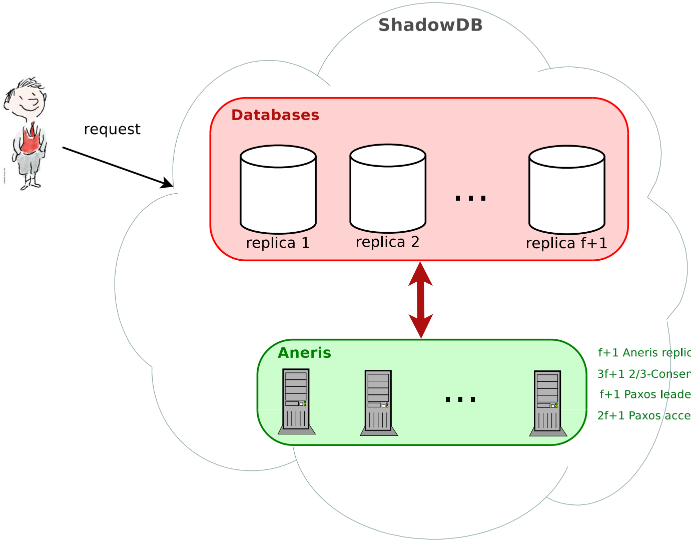
ShadowDB

Aneris: a provably correct ordered broadcast service

Evaluation

Conclusion

The Big Picture



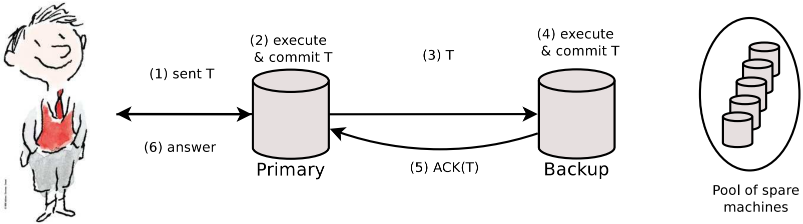
f+1 Aneris replicas

3f+1 2/3-Consensus replicas

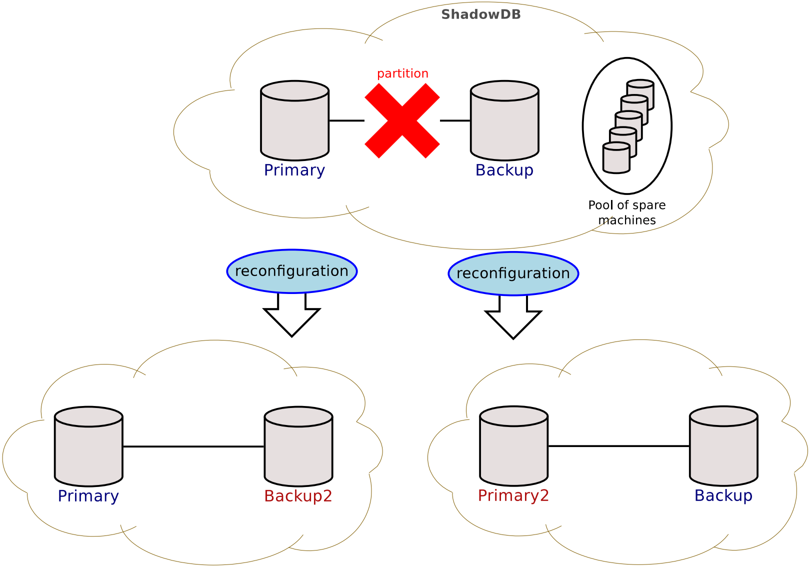
f+1 Paxos leaders

2f+1 Paxos acceptors

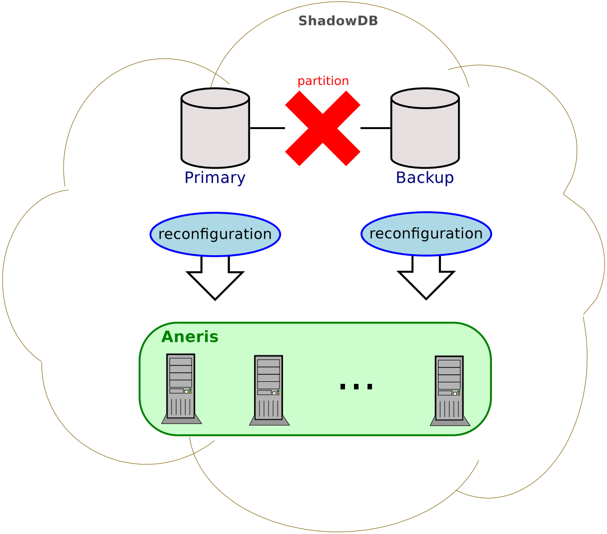
Primary-Backup Replication



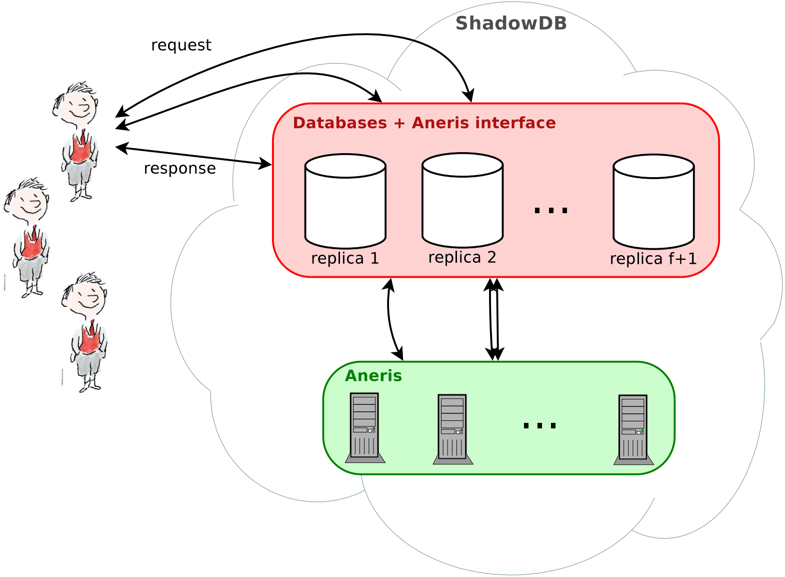
Primary-Backup Replication



Primary-Backup Replication

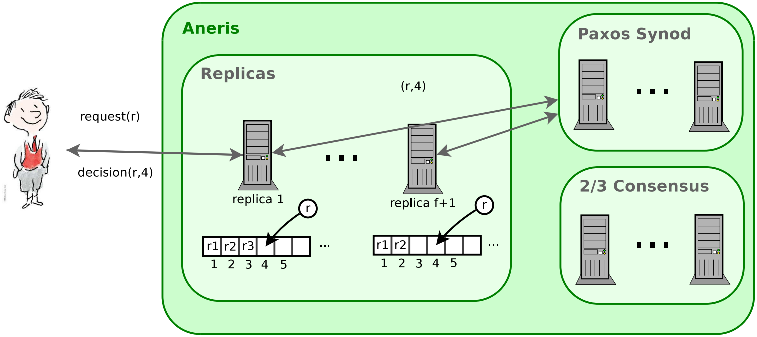


State Machine Replication



Aneris

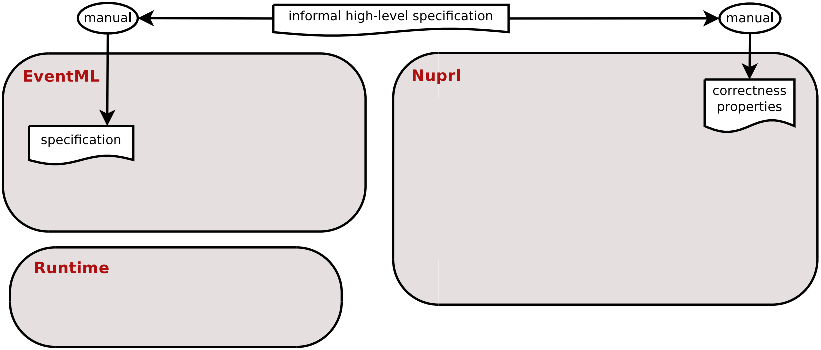
A synthesized and verified ordered broadcast service.



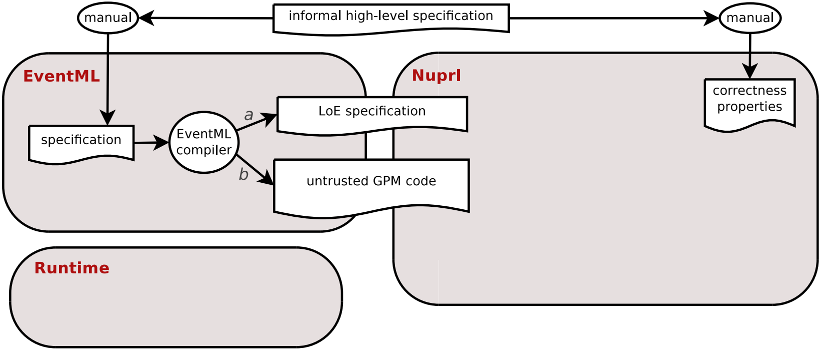
ensures among other things (properties of atomic broadcast):

- **agreement**: for any slot s, if decisions $(r1, s)$ and $(r2, s)$ get delivered then $r1 = r2$.
- **validity**: if decision (r, s) is delivered then r was requested.

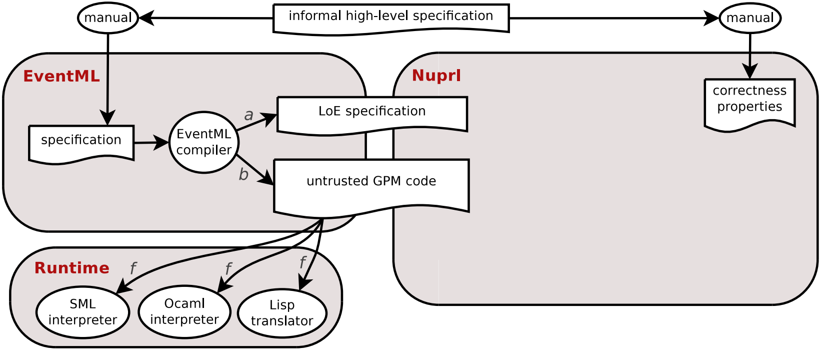
Methodology



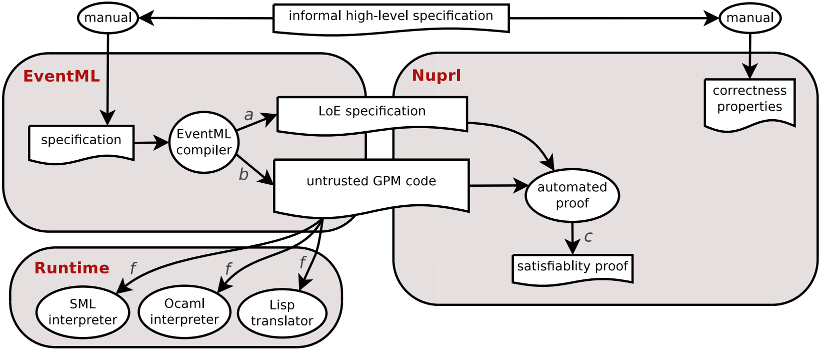
Methodology



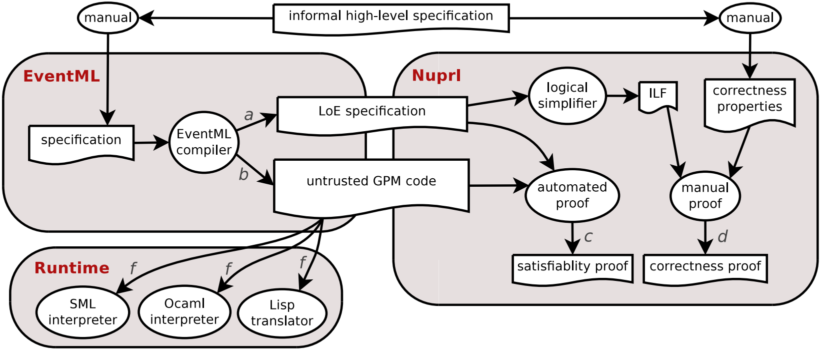
Methodology



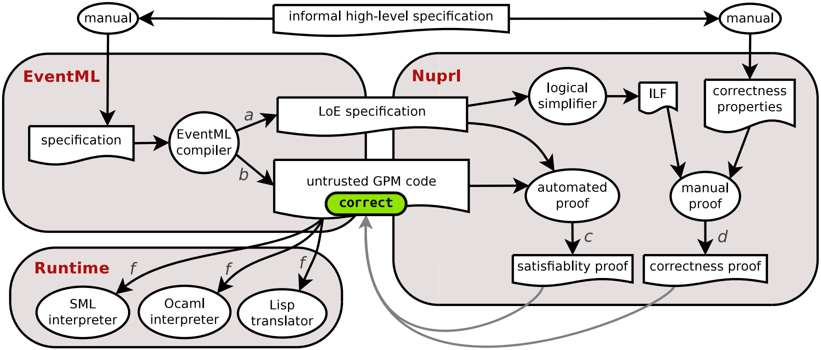
Methodology



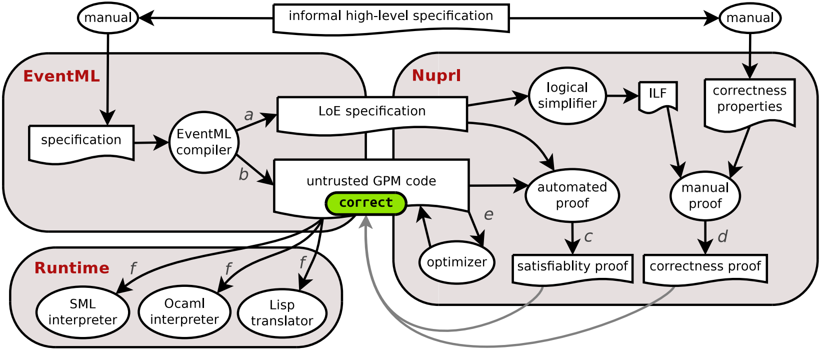
Methodology



Methodology



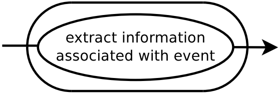
Methodology



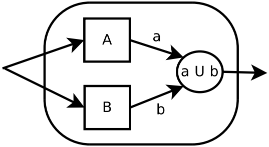
EML, LoE, and GPM

In LoE [BC08, Bic09, BCR12], we specify distributed programs by combining event handlers (similar to Orc) which are all **implementable by simple processes** [BCG10]:

- base:

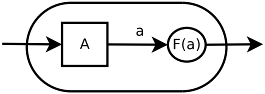


- parallel composition: A || B $\lambda e.A(e) \cup B(e)$

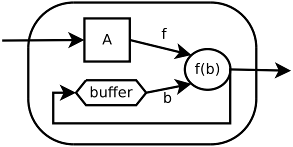


EML, LoE, and GPM

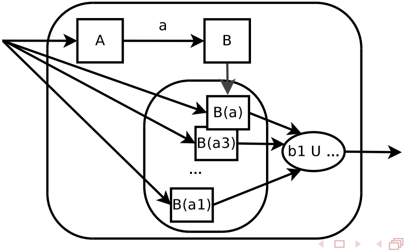
- application:



- buffer:



- delegation:



EventML

2/3-Consensus:

```
..
class TT_Replica = NewVoters >>= Voter;;
main TT_Replica @ locs
```

Paxos Synod:

```
...
class Leader = SpawnFirstScout
             || ((LeaderPropose || LeaderAdopted) >>= Commander)
             || (LeaderPreempted >>= Scout) ;;
main Leader @ ldrs || Acceptor @ accpts
```

Aneris replicas:

```
...
class ReplicaState =
  State(\_.(init_state,{}),
        out_tr propose_inl , swap'base ,
        out_tr propose_inr , bcast'base ,
        out_tr on_decision , decision'base);;
class Replica = (\_.snd) o ReplicaState ;;
main Replica @ reps
```

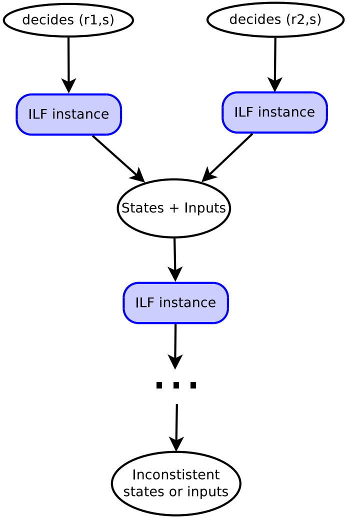
Code Synthesis

Optimized version of the Aneris process:

```
aneris_main-program-opt(Cid;Op;clients;eq_Cid;pax_procs;reps;tt_procs) ==
  λi.case bag-deq-member(λa,b.if a=2 b then inl · else (inr · );i;reps)
     of inl() =>
      fix((λmk-hdf,s.
            (inl (λv.let x,y = v
                     in case name_eq(x;[swap]) ∧b ...
                        of inl(x1) =>
                         let v1 ⟵ ... aneris_propose_inl(Cid;Op;...;...;...;...;...) ...
                         in let x,y = v1 in let v2 ⟵ y @ [] in <mk-hdf <x, y>, v2>
                         | inr(y1) =>
                         case name_eq(x;[bcast]) ∧b ...
                         of inl(x1) =>
                          let v1 ⟵ ... aneris_propose_inr(Cid;Op;...;...;...;...;...) ...
                          in let x,y = v1 in let v2 ⟵ y @ [] in <mk-hdf <x, y>, v2>
                          | inr(y1) =>
                          case name_eq(x;[decision]) ∧b ...
                          of inl(x1) =>
                           let v1 ⟵ ... aneris_on_decision(Cid;Op;...;...;...;...;...;...;...) ...
                           in let x,y = v1 in let v2 ⟵ y @ [] in <mk-hdf <x, y>, v2>
                           | inr(y1) =>
                           let v1 ⟵ s
                           in let x,y = v1 in let v2 ⟵ y @ [] in <mk-hdf <x, y>, v2>) )))
      <aneris_init_state(Cid;Op), []>
      | inr() =>
      inr ·
```

Verification

We use causal induction and inductive logical forms (ILFs).



Verification

E.g., logical explanation of why decisions are made by Paxos:

```
∀[Cmd:{T:Type| valueall-type(T)} ]. ∀[accpts,ldrs:bag(Id)]. ∀[ldrs_uid:Id ⟶ ℤ]. ∀[reps:bag(Id)].
∀[es:EO']. ∀[e:E]. ∀[i:Id]. ∀[p:Proposal].
 (decision'send(Cmd) i p ∈ pax_mb_main(Cmd;accpts;ldrs;ldrs_uid;reps)(e)    decision of p sent to i at e
 ⟺ loc(e) ∈ ldrs                                                          e happens at a leader location
     ∧ (header(e) = ‘‘pax_mb p2b‘‘)                           the decision is triggered by a p2b message
     ∧ (msgtype(e) = P2b)
     ∧ i ∈ reps                                       the recipient of the decision message is a replica
     ∧ (∃e':{e':E| e' ≤loc e }
         ∃z:PValue                                              proposal p is extracted from a pvalue z
          (((header(e') = [propose])          either pvalue z is made from a proposal and current ballot
           ∧ (msgtype(e') = Proposal)
           ∧ ((↑ (proposal_slot (proposal_cmd LeaderStateFun(e'))))
             ∧ (¬↑ (in_domain (proposal_slot msgval(e')) (proposal_cmd (proposal_cmd LeaderStateFun(e'))))))
           ∧ (z = (mk_pvalue (proposal_slot LeaderStateFun(e')) msgval(e'))))
           ∨ ((header(e') = ‘‘pax_mb adopted‘‘)     or either pvalue z received in an adopted message or in leader state
             ∧ (msgtype(e') = pax_mb_AState(Cmd))
             ∧ ((astate_ballot msgval(e')) = (proposal_slot LeaderStateFun(e')))
             ∧ z ∈ map(λsp.(mk_pvalue (astate_ballot msgval(e')) sp);
                       update_proposals (proposal_cmd (proposal_cmd LeaderStateFun(e')))
                                        (pmax(ldrs_uid) (astate_pvals msgval(e'))))))
          ∧ (no commander_output(accpts;reps) z@Loc          this decision is the first output of the commander
               o (Loc,p2b'base(), CommanderState(accpts) (pval_ballot z) (proposal_slot (pval_proposal z)))
            between e' and e)
          ∧ ((pval_ballot z) = (bl_ballot (p2b_bl msgval(e))))
          ∧ ((proposal_slot (pval_proposal z)) = (p2b_slot msgval(e)))
          ∧ ((pval_ballot z) = (p2b_ballot msgval(e)))  the acceptor that sent the p2b message has accepted pvalue z
          ∧ (#(CommanderStateFun(pval_ballot z;proposal_slot (pval_proposal z);es.e';e)) < threshold(accpts))
          ∧ (p = (pval_proposal z)))))          the commander has received a p2b messages from a majority of acceptors
```

Verification

	EventML spec.	LoE spec.	GPM prog.	opt. GPM prog.	correctness properties	correctness proofs
CLK	79N (1H)	590N	452N	249N	73N (1H)	1A/3M (2H)
2/3 Consensus	646N (4H)	1398N	1343N	1752N	122N (1H)	8A/6M (3D)
Paxos-Synod	1729N (2D)	2673N	2625N	3165N	97N (1H)	24A/75M (3W)
Aneris	820N (2D)	1434N	1352N	1245N	418N (1H)	0A/22M (1W)

That was possible thanks:

- to Nuprl's large library of definitions and facts,
- to the powerful **logic of events** theory developed in Nuprl by Mark Bickford and Robert Constable over the past few years (especially to the **delegation** combinator), and
- to the collaboration between the PRL and system groups at Cornell.

Table of Contents

ShadowDB

Aneris: a provably correct ordered broadcast service

Evaluation

Conclusion

Evaluation

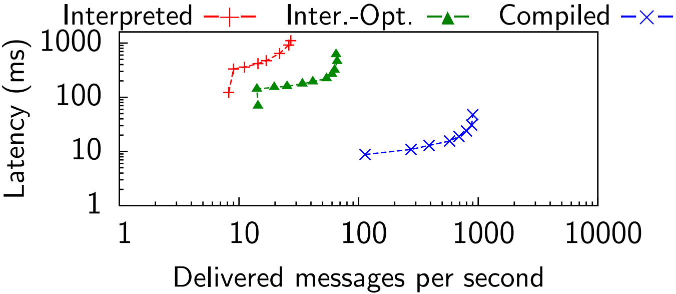
Setup:

- Quad-core 3.6 Ghz Xeons with 4GB running RH 5.8
- Gigabit switch
- Various embedded and in-memory DBs

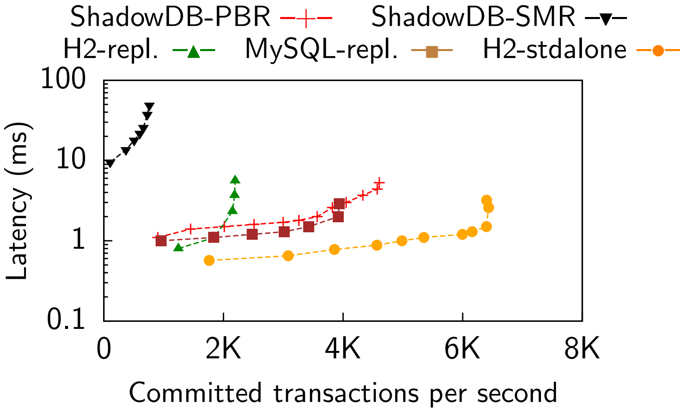
We evaluate:

- Aneris (the broadcast service)
- ShadowDB
  - Micro-benchmark (1 table, single-row update)
  - TPC-C (9 tables, 5 transaction types, 92% updates)

Evaluation - Aneris



Evaluation - ShadowDB - Micro-benchmark



Evaluation - ShadowDB - TPC-C

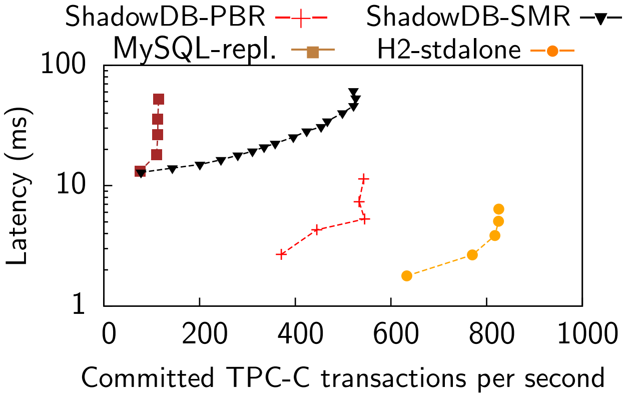


Table of Contents

ShadowDB

Aneris: a provably correct ordered broadcast service

Evaluation

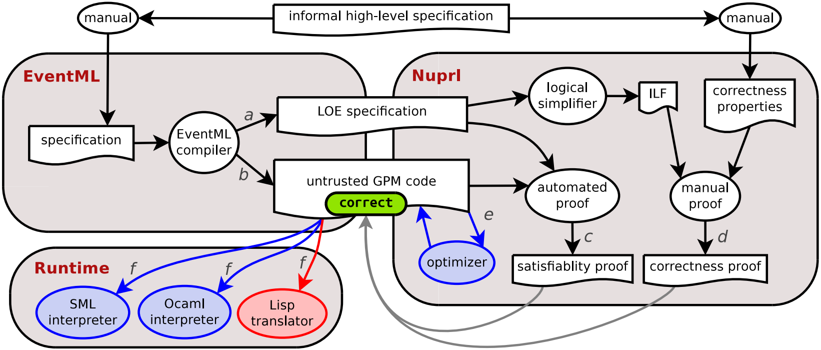
Conclusion

Even More Trustworthy Distributed Systems

Crash-tolerant → Byzantine fault-tolerant

Nysiad

probabilistic systems



informal high-level specification

manual

manual

EventML

specification

EventML compiler

a

b

LOE specification

untrusted GPM code

correct

Nuprl

logical simplifier

ILF

correctness properties

automated proof

manual proof

c

d

e

satisfiablity proof

correctness proof

optimizer

Runtime

f

f

f

SML interpreter

Ocaml interpreter

Lisp translator

Scala interface?

Complexity

Summary

➲ Provably correct distributed protocols.

➲ Aneris in used by the replicated database ShadowDB that itself will be used by Nuprl.

➲ Decent performance.

➲ Example that our methodology to specify (using small human manageable components) and verify (ILFs + causal induction) protocols works.

References I

Mark Bickford and Robert L. Constable.
Formal foundations of computer security.
In *NATO Science for Peace and Security Series, D: Information and Communication Security*, volume 14, pages 29–52. 2008.

Mark Bickford, Robert Constable, and David Guaspari.
Generating event logics with higher-order processes as realizers.
Technical report, Cornell University, 2010.

Mark Bickford, Robert L. Constable, and Vincent Rahli.
Logic of events, a framework to reason about distributed systems.
In *Languages for Distributed Algorithms Workshop*, 2012.

Mark Bickford.
Component specification using event classes.
In *Component-Based Software Engineering, 12th Int'l Symp.*, volume 5582 of *LNCS*, pages 140–155. Springer, 2009.

Vincent Rahli, Mark Bickford, and Abhishek Anand.
Formal program optimization in Nuprl using computational equivalence and partial types.
In *ITP'13*, volume 7998 of *LNCS*, pages 261–278. Springer, 2013.

Vincent Rahli, Nicolas Schiper, Robbert Van Renesse, Mark Bickford, and Robert L. Constable.
A diversified and correct-by-construction broadcast service.
In *The 2nd Int'l Workshop on Rigorous Protocol Engineering (WRiPE)*, October 2012.

Nicolas Schiper, Vincent Rahli, Robbert Van Renesse, Mark Bickford, and Robert L. Constable.
ShadowDB: A replicated database on a synthesized consensus core.
In *Eighth Workshop on Hot Topics in System Dependability*, HotDep'12, 2012.