Specification and verification of consistency models

Paolo Viotti
UPMC - work done at Eurecom

RainbowFS kickoff meeting
May 4, 2017
Consistency: an introduction

The ability of a storage system of maintaining a certain correct state (despite concurrency, partial failures and asynchrony).

- both safety and liveness

• Where
  - Architectures: shared memory, shared disk, shared nothing
  - Databases: relational, NoSQL
    • Transactional and non-transactional
  - From CPU caches to geo-replicated systems

• Tradeoffs
  - CAP, latency
Full stack consistency

Application
Language
Flow
Object
Storage

CISE tool
Bloom, Lasp
Blazes
CRDTs, LVars
Survey, Conver
Outline

**Specifying** consistency

*An axiomatic approach*

[ACM Comput. Surv. ‘16]

**Verifying** consistency

*Conver*

[ACM PaPoC ‘16]
Specifications of consistency models

- State-of-the-art definitions
  - Informal, imprecise
  - Incompatible

- Different contexts
  - Shared-memory systems
  - Databases
  - Outsourced/cloud storage
Consistency specifications:
Operational vs. Axiomatic

Based on ref. implementation

✔ Robust specifications
✔ Refinement mappings to prove implementations correct

✗ Easy to over-specify
✗ Weak models spec. are unwieldy

Based on logical conditions

✔ Composable
✔ Meaningful for users/designers
✔ Concise (even for weak models)
✔ Abstract away implementation

✗ Easy to get axioms wrong
Axiomatic consistency specifications

Consistency semantics as logic *predicates* about *ordering* and *visibility* of events

\[ P_A \rightarrow \text{vis} \rightarrow P_B \rightarrow \text{so} \rightarrow P_C \rightarrow \text{so} \]
A model for axiomatic specifications
refining and extending [Burckhardt ‘14]

• Processes, objects, operations
• Execution ↔ **History** (set of operations)

• Relations on history
  – **rb**: returns-before partial order
  – **ss, so**: eq. relation and partial order on sessions
  – **ob**: equivalence relation on objects

• **Abstract execution** = history, vis, ar
  – **vis**: visibility, tracks propagation of writes
  – **ar**: arbitration total order, how system resolves conflicts
A model for axiomatic specifications
refining and extending [Burckhardt ‘14]

• Operation **context**
  
  − Model state of execution as graph
  
  − Projection on abstract execution

• Return-value consistency
  
  − “Expected” set of return values according to context and…
  
  − …to the replicated data type (set, queue, register…)

• **Consistency models as logic predicates on abstract executions**

\[
H \models \mathcal{P}_1 \land \cdots \land \mathcal{P}_n \iff \exists A \in \mathcal{A} : \mathcal{H}(A) = H \land A \models \mathcal{P}_1 \land \cdots \land \mathcal{P}_n
\]
A survey of consistency semantics

40+ predicates from 30+ years of research
A partial ordering of models
A partial ordering of models
A partial ordering of models
Outline

Specifying consistency

An axiomatic approach

[ACM Comput. Surv. ‘16]

Verifying consistency

Conver

[ACM PaPoC ‘16]
Consistency and the real world

CONSISTENT!*
Verifying consistency:
state of the art

- Strong consistency checkers
  - binary decision problem

- Staleness
  - for eventually consistent clouds

- Precedence graph
  - transactional systems

- Application-level invariant checkers
Verifying consistency: theoretical results

- Linearizability
  - NP-complete (polynomial)* [Gibbons et al., ‘97]
  - Model checking, 1 execution: EXPSPACE [Alur et al., ‘00]

- Sequential consistency
  - Combinatorial problem, 1 execution: NP-complete [Gibbons et al., ‘92]
  - Model checking, 1 execution: undecidable [Alur et al. ‘00]

- Causal consistency [Bouajjani et al., ‘17]
  - Implementation: undecidable (decidable)*
  - 1 execution: NP-complete (polynomial)*

- Eventual consistency*
  - Model checking, 1 execution: EXSPACE-hard [Bouajjani et al., ‘14]
Testing distributed systems

- Traditional testing, distributed tracing, monitoring
  - *Dapper, Zipkin, ...printf( )*

- “Smart testing”
  - Property-based testing, fault injection (*Jepsen*), directed random tests, deterministic simulations

- Formal methods
  - Model checking
  - Correctness-by-construction (*Coq, TLA+...*): *Verdi, IronFleet, Chapar*
  - “Lightweight FM”: invariants verification through SMT: CISE tool
Property-based consistency verification

Verify consistency semantics as axiomatic invariants of executions
Property-based testing

A simple example (in Erlang):

Function to reverse a list

```erlang
reverse([]) -> [];
reverse([X|Xs]) -> reverse(Xs) ++ [X].
```

Property:

for every list Xs, reverse(reverse(Xs)) == Xs

```erlang
prop_reverse() ->
  ?FORALL(Xs, list(int()),
         reverse(reverse(Xs)) == Xs).
```

```
3> proper:quickcheck(qc_test:prop_reverse()).
Ok: Passed 100 test(s).
true
```
Conver: architecture

Test case generation

Targeted | Random

Executions

Fault Injection

WAN Emulation

Verification

OK

Visualization

Consistency Predicates

Targeted Random

OK

Fault Injection

WAN Emulation

Visualization
Conver prototype

- Open source prototype in Scala
  - github.com/pviotti/conver-scala
- Automatic local deployment with Docker
- Can verify 7 consistency models
- 2 data stores (Riak, ZooKeeper)
  - easily extensible
Conver - outputs
Summary

- Declarative/axiomatic specifications of consistency models
  - To reason about and compare them
  - To verify real-world implementations

Future work

- Prove strength relations between consistency models
- Extend Conver
  - Transactional semantics
  - Map application invariants to storage semantics