Cobra: Making Transactional Key-Value Stores Verifiably Serializable

Cheng Tan, Changgeng Zhao, Shuai Mu*, Michael Walfish

NYU Computer Science Department, Courant Institute
*Stony Brook University
Is the database executed correctly?

Dana
Is the database executed correctly?

Dana
Serializability is a correctness contract

- Serializability implies basic data integrity, tolerating faults, ...

\[
W_1(x=1) \quad W_2(x=2) \quad R_3(x):999
\]
Serializability is a correctness contract

gold standard isolation level & correctness contract

serializability violations exist in production systems
The underlying problem is ...

a) ... black-box checking of

b) ... serializability

c) ... while scaling to real-world workloads.

Note: we verify the executions, not the implementation.
Cobra: verifying serializability of black-box databases

Cobra:

- History (clients' transactions)

- A family of graphs

- Acyclic? or none exists?

- This is HUGE
Cobra: verifying serializability of black-box databases

history

set up large search problem

pruning technique 1

pruning technique 2

pruning technique 3

SMT solver

accept/reject

any acyclic? or none exists?
Cobra: verifying serializability of black-box databases

A family of graphs

Any acyclic? or none exists?

History

Set up large search problem → Pruning technique 1 → Pruning technique 2 → Pruning technique 3 → SMT solver

Garbage collection

Accept/reject
Rest of the talk

• The underlying problem

• Solution: Cobra
Are transactions serializable?
Starting point: brute-force search on a polygraph

• [Papadimitriou 79]: build a polygraph (a family of graphs) and search
• Step 1: building a polygraph from a history
• Step 2: searching for an acyclic graph
Set up large search problem

Pruning technique 1 → Pruning technique 2 → Pruning technique 3 → SMT solver

Accept/reject
Starting point: brute-force search on a polygraph

- [Papadimitriou 79]: build polygraph and search
- Step 1: building a polygraph from a history
- Step 2: searching for an acyclic graph

polygraph = (V, E, C)
V = { T1, T2, T3 }
E = { T1 → T2 }
C = { <T3 → T1, T2 → T3> }

\[
\begin{align*}
W_1(x=1) & \\
T_1 & \quad T_2 \\
W_3(x=2) & \\
T_3 & \quad R_2(x): 1
\end{align*}
\]
Starting point: brute-force search on a polygraph

• [Papadimitriou 79]: build polygraph and search
• Step 1: building a polygraph from a history
• Step 2: searching for an acyclic graph

\[ W_1(x=1) \rightarrow T_1 \rightarrow T_2 \rightarrow W_3(x=2) \rightarrow T_3 \]

\[ R_2(x): 1 \] 

T1 \( \rightarrow \) T2 \( \rightarrow \) T3 

acyclic? yes. serializable!
polygraph \((V, E, C)\) 

the search space is \(2^{\lvert C \rvert}\)
Outline

• The underlying problem

• Solution: Cobra
Checking serializability may be tractable in practice

- Intuitions:
  - advances of SAT/SMT solvers
  - heuristically solving many hard problems
- Baseline: encode the problem and use SMT solvers
Cobra aims at real-world workloads

- Intuition:
  - advances of SAT/SMT solvers
  - heuristically solving many hard problems
- Baseline: encode the problem and use SMT solvers

Edges:
\[ e(T_1,T_2) = \text{True} \]

Constraints:
\[ \text{con} = (e(T_3,T_1)=\text{True} \text{ AND } e(T_2,T_3)=\text{False}) \text{ OR } (e(T_3,T_1)=\text{False} \text{ AND } e(T_2,T_3)=\text{True}) \]

Acyclicity:
graph with edges=\text{True} is acyclic
Cobra aims at real-world workloads

• Intuition:
  • advances of SAT/SMT solvers
  • heuristically solving many hard problems

• Baseline: encode the problem and use SMT solvers

slow because of too many constraints
How to reduce constraints in a polygraph?

history → construct polygraph → combining writes → coalescing constraints → pruning → SMT solver (MonoSAT)

accept/reject
Combining writes exploits a common pattern

• Read-modify-write is a common pattern in practice.
Combining writes: exploit common patterns

- Read-modify-write is a common pattern in practice.

Cobra produces just one constraint.
Cobra exploits characteristics of the problem

- history
- construct polygraph
- combining writes
- coalescing constraints
- pruning
- SMT solver (MonoSAT)

accept/reject
Pruning via graph paths (reachability)

• idea: reduce #constraints by reachability

1) what can be learned from reachability?
2) how to get reachability efficiently?
Pruning via graph paths (reachability)

- idea: reduce #constraints by reachability
  1) what can be learned from reachability?
  2) how to get reachability efficiently?

Boolean reachability matrix \((n \times n)\)

\[
T_{ij} = \text{Boolean matrix of 1-step reachability}
\]

\[n = \text{#nodes in the graph}\]
Cobra verifies in rounds to support growing histories

transactions

- construct polygraph
- combining writes
- coalescing constraints
- pruning
- SMT solver (MonoSAT)
- garbage collection
- accept/reject

• Cobra needs to delete transactions after each round.
Experimental evaluation

• What are Cobra verifier's costs compared to the baseline (MonoSAT)?

• How much time is spent on each phase of Cobra?

• What is the Cobra's verification throughput?
• How much runtime overhead does Cobra impose for clients?
• What are Cobra's storage and network costs?
Experiment setup

• Benchmarks:
  • TPC-C, Twitter, RUBiS
  • BlindW: RM (90% reads), RW (50% writes), WM (90% writes)

• Databases:
  • RocksDB, PostgreSQL, and Google Cloud Datastore

• Verifier:
  • p3.2xlarge EC2 instance: a V100 GPU, 8-core CPU, 64GB memory
Cobra can handle 10x larger workloads

- BlindW-RW: read-only and write-only transactions (50:50)
- 10k-key DB, 8 operations/txn, 24 concurrent clients
Decomposition of Cobra's verification runtime

- All workloads are with 10k transactions.
Recap

• Cobra verifies ...
  • ... serializability
  • ... of black-box databases
  • ... while scaling to real-world workloads.
Related work

• Serializability checker for black-box databases
  • algorithms without SAT/SMT [BE19, SMWG11]
  • Gretchen, using a constraint solver (fzn-gecode)

• Elle, an isolation anomaly checker

• Checking/ensuring storage consistency

• Execution Integrity

- black-box checking
- serializability
- scaling to real-world workloads
Related work

• Serializability checker for black-box databases

• Elle, an isolation anomaly checker
  • mode 1: verify serializability by specific APIs and workloads (not black-box)
  • mode 2: testing serializability violations using heuristics (not verification)

• Checking/ensuring storage consistency

• Execution Integrity
Related work

• Serializability checker for black-box databases

• Elle, an isolation anomaly checker

• Checking/ensuring storage consistency
  • Concerto [AEKKMPR17]
  • requiring extra information from the database [RGAKW12, ZK12]
  • relying on synchronized clocks [LVAHSTKL15]
  • requiring client-to-client communication [SCCKMS10]

• Execution Integrity
Related work

• Serializability checker for black-box databases

• Elle, an isolation anomaly checker

• Checking/ensuring storage consistency
  • Concerto [AEKKMPR17]
  • requiring extra information from the database [RGAKW12, ZK12]
  • relying on synchronized clocks [LVAHSTKL15]
  • requiring client-to-client communication [SCCKMS10]

• Execution Integrity
Related work

• Serializability checker for black-box databases

• Elle, an isolation anomaly checker

• Checking/ensuring storage consistency

• Execution Integrity
  • replication: PBFT [CL99], Ethereum
  • attestation: SGX-/TPM-based systems
  • probabilistic proofs: Pepper [SMBW12], Pinocchio [PGHR13], Pantry [BJRSBW13]
  • others: Ripley [VPL09], AVM [HARD10], Verena [KFPC16], Orochi [TYLW17]
Summary

• Cobra verifies serializability of a black-box database ... for real-world workloads.

• Users of cloud databases used to have to assume serializability; but now, with Cobra, they can be sure.

• Code is released at:
  https://github.com/DBCobra/CobraHome