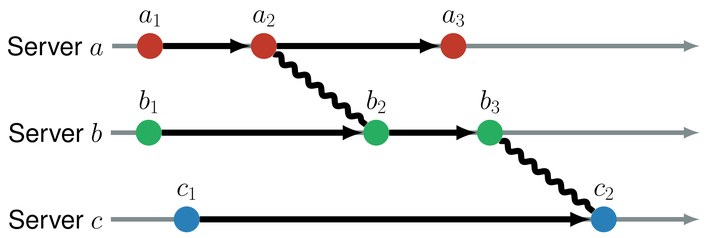
Debugging Distributed Systems with Why-Across-Time Provenance

Michael Whittaker, Cristina Teodoropol, Peter Alvaro, Joseph M. Hellerstein

Causality

Causality applies to arbitrary state machines, but it is an overapproximation of the true causes of an event.



Why-Provenance

Given a relational algebra query Q, a database instance I, and a tuple $t \in Q(I)$, a witness of t is a subinstance $J \subseteq I$ such that $t \in Q(J)$. The **why-provenance** of t, Why(Q, I, t), is the set of minimal witnesses of t.

$$Q \stackrel{\text{def}}{=} \pi_{\texttt{name}}(\sigma_{\texttt{friend1=ecodd}}(\texttt{Users} \bowtie_{\texttt{username=friend2}} \texttt{Friends}))$$

Users

username	name
ecodd	Edgar Codd
jumpman	Michael Jordan
mlpro	Michael Jordan

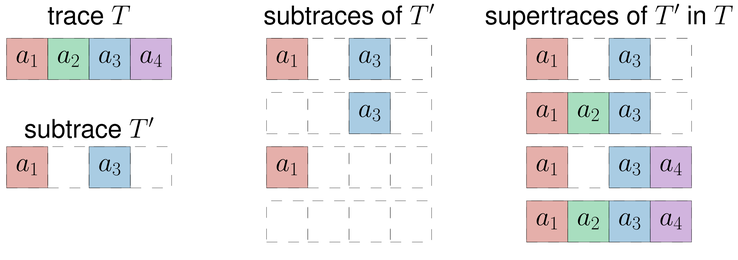
Friends

friend1	friend2
ecodd	jumpman
ecodd	mlpro

Why-provenance returns the precise causes of an event, but it is limited to static relational databases.

State Machines

A **state machine** $M = (S, s_0, \Sigma, \Lambda, \delta, \epsilon)$ consists of a set S of states, a start state $s_0 \in S$, an input alphabet Σ, an output alphabet Λ, a transition function $\delta : S \times \Sigma \to S$, and an output function $\epsilon : S \times \Sigma \to \Lambda$.



Wat-Provenance

Example 1. Consider a key-value server state machine M with an input alphabet that consists of sets and gets to integer-valued variables that are initially 0.

$$\begin{aligned} T &= \text{set}(x, 1); \text{set}(y, 2) \\ i &= \text{get}(x) \\ o &= 1 \end{aligned}$$

Example 2. Consider a state machine M that stores a set of boolean-valued variables that are initially false. Users can set variables to true or false and can request that M evaluate a formula over these variables.

$$\begin{aligned} T &= \text{set}(a); \text{set}(b); \text{set}(c); \text{set}(d) \\ i &= \text{eval}((a \wedge d) \vee (b \wedge c)) \\ o &= \text{true} \end{aligned}$$

Example 3. Consider again the state machine M from the previous example.

$$\begin{aligned} T &= \text{set}(a); \text{set}(b); \text{set}(c) \\ i &= \text{eval}((a \wedge \neg b) \vee c) \\ o &= \text{true} \end{aligned}$$

Given a state machine M, an input trace T, an input i, and the corresponding output $o = \epsilon^*(s_0, Ti)$, we say that a subtrace T' of T is a **witness** of o if $\epsilon^*(s_0, T'i) = o$. We say that a witness T' of o is **closed under supertrace in** T if every supertrace of T' in T is also a witness of o. Let Wit(M, T, i) be the set of witnesses of o that are closed under supertrace in T. The **wat-provenance** of input i with respect to M and T, abbreviated Wat(M, T, i), is the set of minimal elements of Wit(M, T, i).

Example 4. Consider again the key-value server state machine from Example 1.

$$\begin{aligned} T &= a_1 a_2 a_3 = \text{set}(x, 1); \text{set}(x, 2); \text{set}(x, 1) \\ i &= \text{get}(x) \\ o &= 1 \end{aligned}$$

Example 5. Consider a relational database state machine M. The input alphabet of M includes commands to insert a tuple into M and to execute a relational algebra query against M. Initially, all relations are empty.

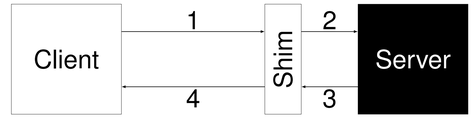
$$\begin{aligned} T &= a_1 a_2 a_3 = \text{insert}(R, t); \text{insert}(R, u); \text{insert}(S, u) \\ i &= \text{query}(R - S) \\ o &= \{t\} \end{aligned}$$

Wat-Provenance Specifications

- Automatically computing the wat-provenance of a black box is infeasible because it requires a complex code analysis of the black box's *implementation*.
- Fortunately, many black box components have simple *interfaces*, even if they have complex implementations.
- A **wat-provenance specification** is a function that—given a trace T and input i—directly returns the wat-provenance of i. Wat-provenance specifications are written against a system's interface (simple) instead of its implementation (complex).
- Examples of black boxes with simple wat-provenance specifications include key-value stores, object stores, distributed file systems, coordination services, load balancers, and stateless services.

Watermelon

- Watermelon is a prototype distributed debugging framework that leverages wat-provenance and wat-provenance specifications.
- Users write wat-provenance specifications in Python or SQL.
- Watermelon uses shims to trace causal history.



Evaluation

System	Language	LOC	Supported API
Redis	SQL	30	get, set, del, append, incr, decr, incrby, decrby, strlen
POSIX	Python	88	reading and writing byte ranges
Amazon S3	Python	200	creating, copying, catting, removing, and listing objects and buckets
Zookeeper	SQL	70	creating, reading, writing, and listing znodes

Debugging Technique	Ease Of Adoption	Runtime Overheads	Supports High-Level Debugging	Supports Low-Level Debugging
SPADE (Audit Logs)	easy	low	no	some
SPADE (LLVM)	easy	high	no	yes
`printf` Debugging	easy–impossible	low	some	yes
Watermelon	easy–hard	medium	yes	no