HAL
open science

The CISE Tool: Proving Weakly-Consistent Applications Correct

Mahsa Najafzadeh, Alexey Gotsman, Hongseok Yang, Carla Ferreira, Marc Shapiro

To cite this version:

Mahsa Najafzadeh, Alexey Gotsman, Hongseok Yang, Carla Ferreira, Marc Shapiro. The CISE Tool: Proving Weakly-Consistent Applications Correct. [Research Report] RR-8870, Inria Paris Rocquencourt. 2016. hal-01279495v2

HAL Id: hal-01279495

https://hal.science/hal-01279495v2

Submitted on 2 Mar 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

The CISE Tool: Proving Weakly-Consistent Applications Correct

Mahsa Najafzadeh
Sorbonne-Universités-UPMC & Inria, Paris, France
Alexey Gotsman
IMDEA Software Institute, Spain
Hongseok Yang
University of Oxford, UK
Carla Ferreira
NOVA LINCS, DI, FCT, U. NOVA de Lisboa, Portugal
Marc Shapiro
Sorbonne-Universités-UPMC & Inria, Paris, France

RESEARCH REPORT
N° 8870
26 February 2016
Project-Team Regal

ISSN 0249-6399 ISRN INRIA/RR--8870--FR+ENG

The CISE Tool: Proving Weakly-Consistent Applications Correct*

Mahsa Najafzadeh
Sorbonne-Universités-UPMC & Inria, Paris, France
Alexey Gotsman
IMDEA Software Institute, Spain
Hongseok Yang
University of Oxford, UK
Carla Ferreira
NOVA LINCS, DI, FCT, U. NOVA de Lisboa, Portugal
Marc Shapiro
Sorbonne-Universités-UPMC & Inria, Paris, France

Project-Team Regal

Research Report n° 8870 — 26 February 2016 — 9 pages

Abstract: Designers of a replicated database face a vexing choice between strong consistency, which ensures certain application invariants but is slow and fragile, and asynchronous replication, which is highly available and responsive, but exposes the programmer to unfamiliar behaviours. To bypass this conundrum, recent research has studied hybrid consistency models, in which updates are asynchronous by default, but synchronisation is available upon request. To help programmers exploit hybrid consistency, we propose the first static analysis tool for proving integrity invariants of applications using databases with hybrid consistency models. This allows a programmer to find minimal consistency guarantees sufficient for application correctness.

Key-words: Weak Consistency; Replicated Database; Invariants; Static Analysis; Application/Database Co-Design

* This research is supported in part by European FP7 FET Young Explorers project ADVENT, and European FP7 Project SyncFree.

RESEARCH CENTRE
PARIS – ROCQUENCOURT

Domaine de Voluceau, - Rocquencourt
B.P. 105 - 78153 Le Chesnay Cedex

L'outil CISE : prouver qu'une application faiblement cohérente est correcte

Résumé : La conception d'une base de données répliquées fait face à un choix difficile, entre la cohérence forte, qui garantit certains invariants applicatifs mais reste lente et fragile, et la réplication asynchrone qui est hautement disponible mais expose le programmeur à des comportements inattendus. Pour dépasser cet embarras, des recherches récentes portent sur des modèles de cohérence hybrides, dans lesquels les mises à jour sont asynchrones par défaut, mais les mécanismes de synchronisation sont disponibles sur demande. Afin d'aider les programmeurs à exploiter la cohérence hybride, nous proposons le premier outil statique capable de prouver les invariants d'intégrité d'applications utilisant une base de donnée à cohérence hybride. Ceci permet au programmeur de trouver les garanties de cohérence minimales garantissant que l'application est correcte.

Mots-clés : Cohérence faible ; base de données répliquée ; invariants ; analyse statique ; conception coordonnée application/base de données

The CISE Tool: Proving Weakly-Consistent Applications Correct

26 February 2016

1 Introduction

To achieve availability and scalability, many modern distributed systems rely on *replicated databases*, which maintain multiple replicas of shared data. Clients can access the data at any of the replicas, and these replicas communicate changes to each other using message passing. For example, large-scale Internet services use data replicas in geographically distinct locations, and applications for mobile devices keep replicas locally to support offline use. Ideally, we would like replicated databases to provide *strong consistency*, i.e., to behave as if a single centralised node handles all operations. However, achieving this ideal usually requires synchronisation among replicas, which slows down the database and even makes it unavailable if network connections between replicas fail [1, 7].

For this reason, modern replicated databases often eschew synchronisation completely; such databases are commonly dubbed *eventually consistent* [14]. In these databases, a replica performs an operation requested by a client locally without any synchronisation with other replicas and immediately returns to the client; the *effect* of the operation is propagated to the other replicas only *eventually*. Unfortunately, this way of processing operations exposes applications to undesirable concurrency behaviours, which may cause bugs such as state divergence or invariant violation [6].

For instance, consider a bank account replicated at different bank branches, which supports operations deposit and withdraw. A programmer would like to ensure an *integrity invariant* that the balance is never negative. Assume that the balance is initially €100. Eventual consistency will allow two users to concurrently withdraw €60 at different branches and thus violate the integrity invariant. To ensure the invariant in this example, we have to introduce synchronisation between replicas, and, since synchronisation is expensive, we would like to introduce it sparingly.

*This research is supported in part by European FP7 FET Young Explorers project ADVENT, and European FP7 Project SyncFree.

To allow this, some research [3, 9, 12, 13] and commercial [2, 4, 10] databases now provide *hybrid* consistency models that allow the programmer to request stronger consistency for certain operations and thereby introduce synchronisation. For example, to preserve the integrity invariant in our banking application, only withdraw operations need to use strong consistency, and hence, synchronise to ensure that the account is not overdrawn; deposit operations may use eventual consistency and hence proceed without synchronisation.

Unfortunately, using hybrid consistency models effectively is far from trivial. Requesting stronger consistency in too many places may hurt performance and availability, and requesting it in too few places may violate correctness. Striking the right balance requires the programmer to reason about the application behaviour on the subtle semantics of the consistency model, taking into account which anomalies are disallowed by a particular consistency strengthening and whether disallowing these anomalies is enough to ensure correctness.

To help programmers exploit hybrid consistency models, we propose the first static analysis tool (called CISE: 'Cause I'm Strong Enough) for proving integrity invariants of applications using replicated databases with a range of hybrid models. Our tool is based on a novel proof rule, which we have proved sound [8]. The tool automates the proof rule by discharging its obligations using an SMT solver. If an obligation fails, the tool provides a counter-example, which the developer can use to understand the source of the problem. Using the tool, we have verified several example applications that require strengthening consistency in nontrivial ways [8]. These include an extension of the above banking application, an online auction service and a course registration system. A demo of the tool is available online [11].

In the rest of this paper, we explain our static analysis by the example of the above banking application.

2 System Model

An application consists of a set of operations Op over some set of objects, and invariants over the objects. The database system consists of a set of replicas, each maintaining a full copy of the database state State.

The replication model uses a Read-One-Write-All (ROWA) approach [5]. A client operation is initially executed at a single replica, which we refer to as its *origin* replica. This updates the replica state deterministically, and immediately returns a value to the client. After this, the replica sends a message to all other replicas containing the *effector* of the operation, which describes the updates done by the operation to the database state. Upon receipt, the replicas apply the effector to their state. Effectors of causally-dependent operations are executed in the same order at every replica; effectors of independent (concurrent) operations are executed in any order.

More precisely, the semantics of operations is defined by a partial function

$$\mathcal{F} \in \mathsf{Op} \rightarrow (\mathsf{State} \rightharpoonup (\mathsf{Val} \times (\mathsf{State} \rightarrow \mathsf{State}) \times \mathcal{P}(\mathsf{Token}))).$$

Given a state $\sigma \in \mathsf{State}$ in which an operation $o \in \mathsf{Op}$ executes at its origin replica, $\mathcal{F}(o)(\sigma)$ determines:

- The return value of the operation, from a set Val. We use a special value $\perp$ for operations that return no value.
- A function defining the *effector* of the operation. This will be applied by every replica to its state: immediately at the origin replica, and after receiving the corresponding message at all other replicas.
- A set of *tokens*, used to introduce synchronisation. We explain them later.

For instance, consider the naïve banking application in Figure 1. A client can read the balance from the local replica, make deposits to and withdrawals from the account, and compute interest, all without communicating with the other replicas. Each operation is associated with a *precondition*—a predicate over the state of its origin replica and parameters that determines when the operation can be safely executed (and the $\mathcal{F}$ function defined). A minimal precondition of the $\mathsf{deposit}(\mathsf{amount})$ and $\mathsf{withdraw}(\mathsf{amount})$ operations is $\mathsf{amount} \geq 0$. Their effectors add amount to (respectively, subtract it from) the balance. The $\mathsf{interest}$ operation's precondition is true and its effector multiplies the balance by the interest rate. (As we will see later, the analysis shows that the precondition of $\mathsf{withdraw}$ needs to be strengthened, and that this effector of $\mathsf{interest}$ is unsafe.)

3 CISE Analysis

3.1 Effector Safety Analysis

The first CISE proof obligation, called the effector safety analysis, verifies that the effector of every operation maintains the invariant when applied to *any* state where the operation's precondition is true (not necessarily the one in which the operation was generated).

Let's try out the effector safety analysis on the simple banking application of Figure 1. According to the analysis, the effectors of $\mathsf{deposit}$ and $\mathsf{interest}$ always maintain the invariant. However, for $\mathsf{withdraw}$, the obligation fails and our tool produces a counter-example: if the balance is zero, a non-zero $\mathsf{withdraw}$ operation makes the balance negative. Therefore, we must fix the issue by strengthening its precondition, so that the amount debited is less or equal than the current balance.

With this correction, the effector safety analysis succeeds. The corrected preconditions are shown at the bottom of Figure 2.

$$
\begin{aligned}
\sigma_{\mathsf{init}} &= 0 \\
I &= (\mathsf{balance} \geq 0) \\
\mathsf{Token} &= \emptyset \\
\mathcal{F}_{\mathsf{deposit(amount)}}(\mathsf{balance}) &= (\bot, (\lambda \mathsf{balance}'.\, \mathsf{balance} + \\
&\quad \mathsf{amount}), \emptyset) \\
\mathcal{F}_{\mathsf{interest()}}(\mathsf{balance}) &= (\bot, (\lambda \mathsf{balance}'.\, (1.05 * \\
&\quad \mathsf{balance}')), \emptyset) \\
\mathcal{F}_{\mathsf{withdraw(amount)}}(\mathsf{balance}) &= (\bot, (\lambda \mathsf{balance}'.\, \mathsf{balance} - \\
&\quad \mathsf{amount}), \emptyset)
\end{aligned}
$$

Precondition	Operation
$\mathsf{amount} \geq 0$	$\mathsf{deposit(amount)}$
true	$\mathsf{interest}()$
$\mathsf{amount} \geq 0$	$\mathsf{withdraw(amount)}$

Figure 1: Simple banking application (incorrect).

3.2 Commutativity Analysis

Effectors of concurrent operations may execute in different orders at different replicas. The second CISE obligation, called the commutativity analysis, checks if all pairs of effectors of such operations *commute*: executing them in any order yields the same result, whatever the starting state.

Let us check this obligation for the specification in Figure 1. Predictably, applying the commutativity analysis proves that deposit and withdraw effectors commute. However, the effector of interest does not commute with that of the other operations, and the tool returns a counter-example. Consider two replicas 1 and 2. The balance is initially €100. Replica 1 is the origin for an interest operation. Replica 2 is the origin for a $\mathsf{deposit}(20)$ operation. Replica 1 first applies the effector of interest and then that of deposit, whereas replica 2 applies them in the opposite order. Depending on the order of execution, the result is different, and the replicas diverge.

We fix this by changing the interest operation to compute the absolute interest at the origin replica and letting its effector add this amount to the local balance of every replica (Figure 2). With this corrected specification, our tool proves that the effector of interest does commute with those of the other operations.

3.3 Stability Analysis

The effector safety analysis verified that that the effector of each operation o maintains the invariant when executed in a state satisfying the precondition of the operation. The precondition holds at o's origin replica, but how do we know that

$$
\begin{aligned}
\sigma_{\mathsf{init}} &= 0 \\
I &= \mathsf{balance} \geq 0 \\
\mathsf{Token} &= \{\tau\} \\
\bowtie &= \{(\tau, \tau)\} \\
\mathcal{F}_{\mathsf{deposit(amount)}}(\mathsf{balance}) &= (\bot, (\lambda \mathsf{balance}'.\, \mathsf{balance} + \\
&\quad \mathsf{amount}), \emptyset) \\
\mathcal{F}_{\mathsf{interest()}}(\mathsf{balance}) &= (\bot, (\lambda \mathsf{balance}'.\, (\mathsf{balance}' + 0.05 \\
&\quad * \mathsf{balance})), \emptyset) \\
\mathcal{F}_{\mathsf{withdraw(amount)}}(\mathsf{balance}) &= (\bot, (\lambda \mathsf{balance}'.\, \mathsf{balance} - \\
&\quad \mathsf{amount}), \{\tau\})
\end{aligned}
$$

Precondition	Operation
$\mathsf{amount} \geq 0$	$\mathsf{deposit(amount)}$
true	$\mathsf{interest()}$
$\mathsf{balance} \geq \mathsf{amount} \geq 0$	$\mathsf{withdraw(amount)}$

Figure 2: Corrected banking application.

it will hold when o's effector is applied at a different replica, which concurrently executes effectors of other operations? The third obligation of CISE analysis, called stability analysis, checks if executing the effector of any other operation o' maintains the precondition of o.

Let us illustrate the stability analysis of the withdraw operation in Figure 1. The precondition of the withdraw operation is stable under the effectors of deposit and interest, but it is not stable under the effector of withdraw. The tool returns the following counter-example. Let the balance be €2. The precondition to $\mathsf{withdraw}(1)$ is verified. However, a concurrent $\mathsf{withdraw}(2)$ (whose precondition is also OK) at a different replica makes the balance zero, now violating the precondition of $\mathsf{withdraw}(1)$. If we were to continue, and to apply the effector of the first withdrawal operation at the second replica, the balance would become negative.

To fix the problem, the developer of the banking application may disallow the execution of withdrawals without synchronisation. To model such concurrency control, we use *tokens* $\mathsf{Token} = \{\tau, \ldots\}$ and a symmetric *conflict relation* $\bowtie \subseteq \mathsf{Token} \times \mathsf{Token}$ between pairs of them. In the banking application, we associate a token τ to withdraw such that $\tau \bowtie \tau$ (similarly to a mutual exclusion lock). This ensures that any two withdrawals synchronise.

Figure 2 presents the corrected banking application, incorporating all the changes outlined above. Our static analysis confirms that this application indeed maintains the integrity invariant.

4 Future Work

In the future, we plan to study proof rules for reasoning about integrity invariants on consistency models weaker than causal consistency. We also intend to automate the analysis of counter-examples in order to generate corrections semi-automatically.

References

[1] D. Abadi. Consistency tradeoffs in modern distributed database system design: CAP is only part of the story. *IEEE Computer*, 45(2), 2012.

[2] Amazon. Supported operations in DynamoDB. `http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/APISummary.html`, 2015.

[3] V. Balegas, N. Preguiça, R. Rodrigues, et al. Putting consistency back into eventual consistency. In *Euro. Conf. on Comp. Sys. (EuroSys)*, pp. 6:1–6:16, Bordeaux, France, Apr. 2015.

[4] Basho Inc. Using strong consistency in Riak. `http://docs.basho.com/riak/latest/dev/advanced/strong-consistency/`, 2015.

[5] P. Bernstein, V. Radzilacos, and V. Hadzilacos. *Concurrency Control and Recovery in Database Systems*. Addison Wesley Publishing Company, 1987.

[6] J. C. Corbett, J. Dean, M. Epstein, et al. Spanner: Google's globally-distributed database. In *Symp. on Op. Sys. Design and Implementation (OSDI)*, pp. 251–264, Hollywood, CA, USA, Oct. 2012.

[7] S. Gilbert and N. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2):51–59, 2002. ISSN 0163-5700.

[8] A. Gotsman, H. Yang, C. Ferreira, et al. 'Cause I'm strong enough: Reasoning about consistency choices in distributed systems. In *Symp. on Principles of Prog. Lang. (POPL)*, pp. 371–384, St. Petersburg, FL, USA, 2016.

[9] C. Li, D. Porto, A. Clement, et al. Making geo-replicated systems fast as possible, consistent when necessary. In *Symp. on Op. Sys. Design and Implementation (OSDI)*, pp. 265–278, Hollywood, CA, USA, Oct. 2012.

[10] Microsoft. Consistency levels in DocumentDB. `http://azure.microsoft.com/en-us/documentation/articles/documentdb-consistency-levels/`, 2015.

[11] M. Najafzadeh and M. Shapiro. Demo of the CISE tool, Nov. 2015. `https://youtu.be/HJjWqNDh-GA`. Video of demo, with explanations.

[12] Y. Sovran, R. Power, M. K. Aguilera, et al. Transactional storage for geo-replicated systems. In *Symp. on Op. Sys. Principles (SOSP)*, pp. 385–400, Cascais, Portugal, Oct. 2011.

[13] D. B. Terry, V. Prabhakaran, R. Kotla, et al. Consistency-based service level agreements for cloud storage. In *SOSP*, 2013.

[14] W. Vogels. Eventually consistent. *CACM*, 52(1), 2009.

RESEARCH CENTRE
PARIS – ROCQUENCOURT

Domaine de Voluceau, - Rocquencourt
B.P. 105 - 78153 Le Chesnay Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399