#112 Data Consistency in Transactional Storage Systems: A Centralised Semantics

Email notification
Select to receive email on updates to reviews and comments.

PC conflicts
None

Submitted

Submission ⓘ 11 Jan 2020 12:00:05am AoE · f713d345

▶ Abstract

We introduce an interleaving operational semantics for describing the client-observable behaviour of atomic transactions on distributed key-value stores. Our semantics builds on abstract states comprising centralised, global key-value stores and partial client views. We provide operational definitions of consistency models for our abstract states which we show to be equivalent to the well known declarative definitions of consistency models on abstract

▶ Authors (blind until review)

S. Xiong, A. Cerone, A. Raad, P. Gardner
[details]

Appendices

appdix.pdf

Supplementary Material

tech-report.pdf

▶ Topics

	OveMer	RevExp
Review #112A	B	2
Review #112B	A	3
Review #112C	C	3

2 Comments: đĠŸŷŧ ÕŸĠ (S. Xiong); Reviewer C

You are an **author** of this submission.

Edit submission

Reviews and comments in plain text

Review #112A

Overall merit

Reviewer expertise

B. OK paper, but I will not champion it. **2.** Some familiarity

Paper summary

The paper introduces a new operational, interleaving model of atomic transactions on distributed key-value stores. What is novel in this work is that the operational model enables both (a) verifying that database protocols satisfy specific consistency models, and (b) proving invariant properties of client programs with respect to consistency models. The verification framework is evaluated by means of two case studies: the COPS protocol (for fully replicated key-value stores) and the Clock-SI protocol (for partitioned key-value stores). Furthermore, it is shown how the model can be used to prove invariant properties of transactional libraries, showing, e.g., that a library is robust against a certain consistency model.

Strengths

+ The study of transaction semantics and consistency models has seen renewed interest in the context of highly available distributed systems that need to provide low latency to potentially geo-distributed clients. Thus, the paper addresses an important and timely topic.

+ The existing literature on consistency models has been relatively weak in terms of operational models, with much work based on other models such as execution graphs or dependency graphs. Depending on the context, an operational semantics has a number of advantages, however; thus, the present paper makes an important contribution towards closing a hole in the literature.

+ The formalization is carefully developed; definitions and notations have been chosen carefully. Proofs of theorems are provided either in a companion technical report or as proof sketches in the paper. Thus, the framework appears to be sound.

Weaknesses

– The presented formal framework has not been applied in the construction of practical tools. Thus, some of the potential benefits of the model remain unclear.

– As far as I can see there is no mechanization of the presented formal development.

Comments for authors

The study of transaction semantics and consistency models has seen renewed interest in the context of highly available distributed systems that need to provide low latency to potentially geo-distributed clients. At the same time, existing formal models for consistency models have been limited to specific purposes (e.g., the verification of distributed protocols). The present paper contributes a new operational semantics and a novel way to define consistency models for transactions by means of so-called execution tests.

Section 3:

Lines 315-316: you write: "We require that a client view be atomic in that it can see either all or none of the updates of a transaction."

It is clear that a client view should never see only some of the updates of a transaction; however, in

which situation/model would it be OK if a client view sees none of the updates of a transaction?

Line 346: it is common to call it "core language" or "core calculus" instead of "programming languages" when formalizing such a language.

Lines 363-364: you write: "lookup x := [E] and mutation [E] := E used for reading and writing a single key to kv-stores respectively, which can only be invoked as part of an atomic transaction."

I must say that this notation is confusing: at first, I thought that the [E] in x := [E] is a nested transaction, because you use square brackets for transactions in the definition of C. However, now, the slightly smaller square brackets are used for lookup and mutation, respectively. I strongly suggest to improve the notation.

Line 461: I was a bit surprised to read about related work at this point. I believe this part on related work should be moved elsewhere.

Section 4:

I find the introduction of a new consistency model (WSI) very interesting. It demonstrates the utility of the operational semantics and execution tests for defining consistency models.

Section 5:

Lines 803-805: has the notion of robustness that you are using been used in previous work? If so, please provide a reference, since it is not clear whether you are introducing that robustness notion or not.

Typos:

Line 23: "unified definitions of consistency model" -> "unified definitions of consistency models"

Review #112B

Overall merit

A. Good paper. I will champion it at the PC meeting.

Reviewer expertise

3. Knowledgeable

Paper summary

Summary

The paper describes an operational semantics for distributed transactions with different consistency models. The semantics is based on a centralized multi-version key-value store. The distribution of the store is modelled by client views, such that each client can see different information regarding the store. The approach enables detecting anomalies, e.g., Lost Update, that can occur with weak transactional isolation.

The paper introduces a sequential language for programs and transactions. The operational semantics defines how transactions affect the store. The semantics depends on execution tests which are used to specify consistency models. The definition of execution tests breaks down to two predicates -- `vshift` and `can-commit`. `vshift` describes how a store/view changes when a transaction is applied, and `can-commit` describes whether a transaction can commit in some specific state. The authors show that many common consistency models, e.g., Monotonic Reads, can be modelled with execution tests.

The applicability of the semantics is shown by verifying that (a) the COPS database satisfies causal consistency, and (b) several database libraries are snapshot isolated.

Strengths

Assessment

The paper tackles the a relevant problem: formalizing a store that supports different consistency models. The paper introduces the proposed formal model in a clear way. The approach is explained well, both formally and informally. The applicability of the formalism has been shown sufficiently.

I think the paper would be a good contribution to ECOOP2020.

Weaknesses

The paper says that resolution policies other than last-writer-wins can be "straight-forwardly" modelled with your semantics (L.339). However, to model a sensible resolution policy, a notion of concurrent updates is required, as most resolution policies require to know which updates to merge. How would you detect concurrent operations in your formalism?

Can you model aborts and rollbacks in your semantics? If so, how do you handle assignments to local variables in an aborted transaction?

The definition of transactional snapshots is not clear (Definition 5). While it is hinted to in the running text, just from the formal definition I see no difference to normal snapshots.

It would be interesting to see how availability affects the definitions of consistency models. Is it possible to infer the availability of a consistency model from the definition of execution tests, e.g., by looking at which parts of the store are used in the execution test?

Comments for authors

Minor

- Something went wrong in the caption of Figure 1
- L.109: "libaray" -> "library"
- L.111: "We believe out" -> "We believe our"
- L.228: Merge conflict
- L.510: "last write wins" -> "last-writer-wins" (stay consistent with the spelling)
- L.518: line too long
- L.756: "5) cl reads (R, k2 , (t2 , r2)) from r2 ;" Doesn't cl still read from r1?
- L.826: "mulit-counter" -> "multi-counter"

Review #112C

Overall merit

C. Weak paper, though I will not fight strongly against it.

Reviewer expertise

3. Knowledgeable

Paper summary

The paper introduces a state-based formal framework for describing client-observable behaviour of atomic transactions on distributed key-value stores. The paper describes the operation semantics based on abstract states which employs the concepts of a centralized store, client views and execution tests (implementing the consistency models). This semantics is used to verify distributed protocols like COPS, and invariants on client programs.

Strengths

-The paper describes how different consistency models can be expressed in the semantics as execution tests.

-The paper shows that the formalism can be used to explore new consistency models, such as Weak Snapshot Isolation (WSI).

-The paper shows applications of the operational semantics; in particular, it shows how to map COPS protocol and application invariants from several client programs (a distributed counter, banking library) to a consistency model.

Weaknesses

-The paper suffers from writing issues including unclear statements, and claims which are not properly supported (see detailed comments).

-The paper does not provide a convincing motivation for a new operational semantics for distributed databases.

-The paper does not clarify the contributions of the formalism. Crooks. et al.[18] have proposed a similar state-based formal framework, which can model a superset of the consistency models expressed in the execution tests, and has also been applied to a replicated transactional key value store.

-It is unclear if the WSI model is a contribution in the context of an actual implementation, or if it is just relevant to show that the formalism can express intersections of existing consistency models.

Comments for authors

-The introduction fails to precisely convey what the current state of the art lacks to motivate the need for a new formalism. Even though it discusses the closest related work [16,26,35], there are too many unclear or bold statements. Examples include: -Line 29 states that there has been no previous work on distributed transaction that is "suitable for verifying implementations and for reasoning about clients". However, [18] seems to have studied this and proposed a similar approach to this work. -Lines 58-60 criticize [18] for not being suitable for analyzing client programs, but the formalism in [18] is also based on application observable states and set of

consistency models it can express is larger than the one in this paper. The introduction should clarify what makes [18] not suitable for reasoning about clients programs. -Line 73 claims that it is unlikely for [35] "to be suitable for invariant-based analysis associated with the state" without evidence or argumentation. -Line 115 claims that the semantics provides an interesting mid-point between distributed databases and clients, but it is unclear what a mid-point means.

-Since the work assumes that transactions satisfy the snapshot property and a last-write-wins resolution, the proposed formal framework cannot express consistency models expressed in related work ([18]) like read committed, read uncomitted. It is unclear, however, why the assumptions are necessary, and why they are not limiting the applicability of the approach to distributed key-value stores.

-Although the proposed model is less expressive than [18], it may have some potential since the reduction steps only require the states of all clients, instead of the list of all client operations. Unfortunately, the paper does not show that this new strategy yields benefits. It would make the paper stronger if it provided some evidence of benefits, e.g by showing the the verification is more efficient, or that it allows to prove new properties, or prove existing ones more easily, etc.

-The explanation of Figure 1 is unclear. Either the increments happen concurrently in which case client2 gets the view shown Figure 1c and then last write wins chooses one of the updates, or the increments are causally related and as such client2 gets the view shown in Figure 1d. However, the texts seems to indicate that client2 could choose which views will get.

-Could you add some reference to support the statement in lines 243-244 that the protocols have only been shown to satisfy specific consistency models defined for particular implementations? It would be also useful to make explicit why it is a problem to only verify a protocol in an implementation rather than abstractly in an operational semantics (as implied by the text).

-Line 245 talks about proving invariants of client programs by mapping them to consistency models. It is unclear why counter or a banking application is called a library. Is it because it is an application which could be used by client code?

-Line 307 defines well-formed according to the snapshot assumptions and those in 3.2 and 3.3, while Definition 2 seems use well-formed for the partial order of updates observed by a client. It may be better not to employ the same term, or clarify the relation between them if it is meant as the same property.

-Line 338 claims that the formalism can work with other resolution policies than last-write-wins, but there is no evidence that this could be changed.

-Line 481 indicates that eventual consistency leads to "good performance but anomalous behaviour" without further argumentation. Which kind of anomalous behaviour is the paper referring to? Is it about the lost update anomaly? Sequential Eventual Consistency (SEC) [1] has proven to guarantee convergence (i.e. two replicas will eventually converge to the same state without conflicts once they receive the same updates) and as such, CRDTs like counters do not suffer from such a problem.

-It is unclear what the largest execution test refers to in line 503.

-The paper should explicitly discuss which assumptions are made for distributed communication (ie. which guarantees are assumed on the communication of updates between the clients and the store?). I believe the paper assumes reliable causal order broadcast but it is never made explicit.

-Section 5.2 lacks structure and it could be improved by explicitly explaining how the different "proofs" contribute to the overall storyline or to validate contributions of the approach. The proofs provided boil down to map some application or protocol to a consistency model using execution tests. That seems to be orthogonal to demonstrating that the protocol/application satisfies the consistency model. Since [18] features a concept similar to a execution test it should be then possible to prove the same properties than this work. It would make the paper stronger if the subsection "Invariants vs. Execution Graphs", also discusses why this semantics can help to establishing those invariants more easily than commit tests in [18].

Minor presentation issues:

Line 11: typo in "We believe out" . Line 228: broken text. Line 471: the term strong consistency is overloaded in this sentence. Causal consistency is usually considered a subtype of strong consistency. Table 6: what are the two columns showing? a boolean ? a set?

References used in the review:

[1] M. Shapiro, N Preguiça, C. Baquero, and M. Zawirski. 2011. A comprehensive study of Convergent and Commutative Replicated Data Types. Technical Report 7506. INRIA

Response

Author [Shale Xiong] 25 Mar 2020 **1105 words**

We thank the reviewers for their comments. We will work hard to improve the presentation of the paper and act on all the detailed comments. We answer the main comments first and some of the more detailed comments later.

Main Comments

Reviewer 1: mechanisation and tools

This is a large undertaking that can only be done now that theory has been achieved. In fact, we are working on implementing our semantics, using it to generate litmus tests and prove robustness results for client programs.

Reviewer 2: concurrent update with merge

Our aim was to capture an interleaving operational semantics capturing atomic visibility, since a large number of implementations of distributed systems, including COPS and Clock-SI, are specifically designed to capture the atomic visibility of transactions. We agree that it is interesting to look at concurrent updates with a merge strategy. This would require changing the core programming language and update rule, but probably not the state. We will clarify.

Reviewer 2: aborts and rollbacks

We assume the reviewer is asking about aborts associated with an execution test failing. This is straightforward, by adapting the update to augment the values with abort information and using transaction-local variable stores rather than thread-local variable stores, to prevent information leaking outside aborted transactions. We can capture rollbacks similarly by changing the update rule in the operational semantics. While these adaptations would enable us to express a greater range of consistency models, e.g. opacity, we felt that this would have made the paper more difficult to read. We can comment on this.

Reviewer 3: related work

The reviewer asks for an in-depth comparison between our work, and [18,35]: [18] is a trace-based semantics; [35] is a graph-based semantics; and ours is a state-based semantics. All approaches have their merits.

All assume the last-write-wins policy. [35] and this paper assumes snapshot properties, because the focus is on weak consistency models widely used in distributed databases. [18] does not assume snapshot properties, because the focus is on isolation levels such as Read Committed. We agree that [18] captures more consistency models than this paper.

In [18], the authors show that several definitions of SI collapse into one. They do Ör Âverify protocol implementations and do Ör Ãprove invariant properties of client programs. We have verified protocol implementations and proven invariant properties of client programs. We believe [18] can be used to verify implementations. We believe it is difficult to use [18] to prove invariant properties of client programs. Their notion of trace includes a large amount of information (for example, the total order on transactions) that just would not be observable by a client (for example, a client on one replica does not necessarily see a transaction on another replica). In our opinion, it is not clear that the approach of [18] is appropriate for client reasoning. Our concern is shared also by [26]:

> However, they do not consider verification (manual or automated) of client programs, and it is not immediately apparent if their specification formalism is amenable for use within a verification toolchain.

[35] proposed a fine-grained operational semantics on abstract exactions and developed a model-checking tool for the violation of robustness. This was achieved by converting abstract executions to dependency graphs and checking the violation of robustness on the dependency graphs. The approach has two issues. First, despite [35] assuming atomic visibility of transactions, it presents a fine-grained semantics at the level of the individual transactional operations rather than whole transactions, introducing unnecessary interleavings which complicates the client reasoning: for example, increasing the search space of model-checking tools. In contrast, our semantics is coarse-grained in that interleaving is at the level of whole transactions. Second, all the literature that performs client analysis on abstract executions achieves this indirectly by over-approximating the consistency-model specifications using dependency graphs [9,14,15,16,35]. It is not known how to do this precisely [16]. In contrast, in our work we prove robustness results directly by analysing the structure of kv-stores, without over-approximation. We also give precise reasoning about the mutual exclusion of locks, which we believe will be difficult to prove using abstract executions.

Reviewer 3: the WSI consistency model

The WSI consistency model in this paper has been used to establish a proof technique for proving the robustness of libraries of a certain general form against WSI and hence SI. It has not been justified with respect to an implementation which is why we did not emphasise it in the introduction. We will clarify.

Detailed Comments

Reviewer 1

- Line 315. In a replicated database such as COPS, one client can update one replica and a client on the other replica might see none of the updates.

Reviewer 2

- Definition 5. This is a formal definition of a normal snapshot.
- Availability. We are not sure if we can capture availability in our operational semantics. This is worth investigation.

Reviewer 3

- Line 115. We regard our operational semantics as an interface, or mid-point, between implementations and clients. We will clarify.
- Fig 1. We mean that it is possible for the client to have either the view in (1c) or the view in (1d) under CC. It can only have the view (1d) under UA, PSI and SI. We will clarify.
- Line 243-244. In [33] and [22], the correctness of the COPS and Clock-SI protocol implementations is given by appealing to specific definitions of consistency models that are dependent on these particular implementations. In contrast, we have proved that the protocol implementations are correct by appealing to our general definitions of consistency models that are independent of the particular implementations.
- Line 245. Yes, we are thinking of a library as an application that an be used by client code.
- Line 338. We will take out this claim about modelling resolution policies other than last-write-wins. It needs better justification. (We were thinking of multi-value registers where clients is able to choose a value from the whole view.)
- Line 481. We mean the anomalous behaviour allowed by the weak consistency models, such as the examples given in Figure 7. We will clarify.
- Line 503. The largest execution test means the largest execution test possible in our semantics: that is, the can-commit and vshift properties correspond to true.
- Reliable causal order broadcast. Our framework abstracts from communication between replicas, hence it does not assume reliable causal order broadcast (RCOB). However, the encoding of COPS into our framework is made easier by the fact that the protocol relies on RCOB. We agree that verifying a protocol that does not rely on RCOB in our framework would be a much more difficult task, and we will look to implementations of such protocols in future.

@**A1** Reviewer C 8 Apr 2020

PC Discussion Summary

We thank the authors for their replies which have clarified important concerns and have helped the online discussion. We would like to see the clarifications from the author response in the final version of the paper. We also suggest authors to include the following change:

-Update the introduction to concisely motivate the work based on the state of the art and then move to a dedicated related work (sub)section the in-dept discussion on related work including the clarifications from the author response. This (sub)section could be also used to place related work discussion from line 461.