MVFS:a Framework of Consistency Checking in Distributed System \*

ABSTRACT

Nowadays more and more Internet-scale systems replicate data in distributed data centers for better performance. However, replicas bring us both convenience and the issue of conflict resolution. In most scenarios, distributed system designers have to sacrifice consistency for availability and partition tolerance. Therefore, some consistency models have been proposed for tradeoffs between consistency and performance. In our research, we investigate the problem of checking whether a given execution trace of a distributed data-store system adheres to a certain consistency model. And we design a scalable framework called minimum-visibility-first searching which can give the traces produced by different distributed systems an efficient check upon different consistency models. Moreover, we devise some techniques for the optimization of checking and a suite of supporting tools.

CCS CONCEPTS

• **General and reference → Validation;** • **Software and its engineering → Consistency;**  **Testing**

KEYWORDS

weak-consistency, testing, distributed system

ACM Reference format:

FirstName Surname. 2020. MVFS: a Framework of Consistency Checking in Distributed System.

1 Introduction

Nowadays more and more Internet-scale systems replicate data in distributed data centers for large throughputs, low latency and high fault-tolerance. Just as there is no free lunch in the world, replicas bring us both the improvement of performance and the issue of conflict resolution which is a rather thorny problem.

In 2000, Eric Brewer introduced the idea that there is a fundamental trade-off between consistency, availability and partition tolerance, which has become known as the *CAP Theorem* [1]. Since it is impossible to achieve both consistency and availability in a system with unreliable network, it is necessary to sacrifice one of these desired properties [2]. Partition tolerance is indispensable in the practical scene because the system must continues to operate despite an arbitrary number of messages being dropped or delayed by the network between nodes. Also we cannot sacrifice availability considering the user experience that every request receives a non-error response in a certain time in spite of no guarantee that it contains the most recent write. Hence, programmers have no choice but to use weak consistency to provide specifications instead of strong consistency.

Eventual consistency is a popular consistency model with the development of Conflict-free Replica Data Types, a distributed data types committing to low latency. However, eventual consistency is too ambiguous and many distributed storage systems which claim satisfy eventual consistency are actually stronger than eventual consistency. We should take out a more precise measurement for specifying these systems. So many consistency models such as basic eventual consistency, causal consistency, sequential consistency, have been proposed to help specify and verify distributed systems especially distributed data storage. Different consistency models have different guarantees, different performance and different difficulties of implementation. Given the potentially-huge amount of systems that rely on these distributed data storage systems, it is important to maintain precise specifications and ensure that implementations adhere to their consistency specifications.

Testing a distributed data storage system raises two issues. First, it is hard to derive a suitable set of testing scenarios, e.g., faults to inject into the system and the set of workloads to be executed. Second, it is hard to check whether the given execution satisfies the consistency model efficiently. The Jepson framework has been design to solve the first issue by using randomization [3]. However, the second issue is ignored in some sense. There are some consistency models which specify the weak-consistent distributed data storage, but we find few validation tools can check real executions upon various consistency models due to the complexity of checking correctness.

In this work, we aim to develop a platform called minimum-visibility-first searching that checks the executions from different systems upon different consistency specifications under the visibility-arbitration framework in order to get a precise specification of a system [4]. Moreover we use pruning technique to reduce the searching space of checking, which makes it possible to check large-scale execution traces in an acceptable time. With these tools, we can obtain the relatively precise consistency specification of the system and find the subtle difference on consistency between implementations of the same algorithm thus we can measure the consistency of distributed data storage systems.

We plan to provide an experimental evaluation of our platform on the executions of Riak, which claims to implement a set of conflict-free replica data types. We also want to check the executions of Cassandra, which can change its consistency through the setting of quorum.

2 Related Work

Burckhardt gives us the specification methodology which uses visibility relations and a formalization of consistency criterion [4]. The visibility relation represents the fact that an operation observes the effects of another operations. Emmi and Enea develop a simple annotation language for specifying weak-consistent operations in Java concurrent objects via visibility relaxation, which also naturally capture consistency mechanisms in the distributed systems and develop a validation methodology for specifying software whose operations satisfy multiple distinct consistency levels [5]. Their relaxed-visibility specification has a more powerful expression ability than Burckhardt's along a few different axes, which can specify the weak consistency on replica and message-passing.

Biswas and Enea have done a great work on checking transactional consistency which inspires us a lot [6]. However, this work focuses on consistency models for transaction of modern databases while our research focuses on consistency models for weak-consistent replica data storage such as conflict-free replica data type. In other words, Biswas and Enea aim to check consistency models like read committed, read atomic, and causal consistency while we aim to check consistency models like basic eventual consistency and weak consistency via visibility relaxation.

Chao Wang and Enea address the problem of specifying and verifying CRDTs by introducing a new correctness criterion called Replication-Aware Linearizability which is inspired by Linearizability, but they have not use it to verify real client applications of CRDTs [7].

Michael Emmi have developed the first completely-automatic algorithm for checking weak consistency of concurrent object implementations and identified an optimization to weak-consistency checking [8]. Since this algorithm is designed to check java concurrent objects, we want to use its framework to implement a platform to check distributed systems.

3 Problem Formulation

If we put a workload into a real distributed system, we can get a log from the system. A log file is usually a sequence which contains events of request arguments and events of reply results. We can obtain orders between some events from timestamps or causality of message-passing. However, logs of different systems vary from each other, so we must extract key information from logs in order to obtain the abstraction of logs.

We can define a set of observable behaviors called histories. A history records all the interactions between clients and the system. We include the following information in each history: The operations performed; Whether the operation completed, and what value was returned; The relative order of non-overlapping operations; The session an operation belongs to. Formally, we use event graphs to represent histories. A history is an event graph (*E*, *op*, *rval*, *rb*, *ss*) where *op* describes the operation of an event; *rval* describes the value returned by the operation; *rb* is the returns-before order, a natural partial order on *E*; *ss* is the same-session order, a equivalence relation on *E*.

In order to justify a history, we can add *visibility* and *arbitration* relations which allow us to define not just linearizability, but the whole spectrum of consistency models for eventually consistent systems. *Visibility* is an acyclic relation which tells us about the relative timing of update propagation and operations. If an operation *a* is visible to *b*, it means that the operation *b* have observed the effect of the operation *a*. *Arbitration* is a total order on operations which indicates how the system resolves update conflicts. If an operation *a* is arbitrated before *b*, it means that the system considers the operation *a* to happen earlier than operation *b*. So we can formally define an abstract execution as an event graph (*E*, *op*, *rval*, *rb*, *ss, vis, ar*) where (*E*, *op*, *rval*, *rb*, *ss*) is a history; *vis* is an acyclic and natural relation; *ar* is a total order.

As giving the formal definition of history and abstract execution, we finally can define consistency models. Additionally we have to introduce a definition called consistency predicate. A consistency predicate is a predicate or property of an abstract execution. A consistency model is a collection of consistency predicates [3].

The checking problem is to find an abstract execution that satisfy the consistency model for the given history or make sure there is no correct abstract execution.

4 Methodology

We plan to design the platform using Java. First, we design a collection of classes which store the necessary information about logs, histories and abstract executions. Considering different systems, we provide a set of interfaces that can change log files into the same format.

For the convenience of processing, we organize all the operations and relations as a directed acyclic graph and use a backtracking algorithm to search an abstract execution that satisfies the given consistency model. The reason we name the platform minimum-visibility-first searching is that the algorithm traverses the directed acyclic graph considering only the minimal visibility relations which adhere to the various constraints of the given criterion rather than enumerating every possible visibility relation . For better performance, we want to use pruning to skip redundant searching space.

5 Preliminary Results

We have already developed a set of tools that extract histories from real log files. We also have developed a prototype which can check weak-consistency based on visibility relaxation.

Moreover, we obtain some real logs files from CRDT-Redis, a CRDT built in Redis [9]. We successfully checked these logs which have 9 sessions and more than 20 operations whether satisfy the Replica-aware Linearizability within several minutes.

6 Discussions

In fact, the enumeration of linearizations of an execution’s operations and the enumeration of possible visibility relations among the linearized operations are both exponential. Though the optimization of minimal visibility can help us skip unnecessary enumerations, the cost of checking algorithm is still considerable.

Therefore, we have to assume fixed bounds for certain parameters of the input executions such as the number of threads and the maximum number of overlapping operations to handle real long executions.

7 Conclusion

Our platform that integrates a set of checking techniques and supporting tools can check whether an execution of a distributed system satisfies a given consistency model. Our platform enables programmers to give a precise specification to a weak-consistent distributed system and verify the correctness of the implementation. Our platform is flexible enough to handle the executions that come from different systems upon different consistency models. Moreover, our platform is open to users’ further modification.

We also have experiments on Riak and some other CRDTs, which show the performance of our checking algorithm.

ACKNOWLEDGMENTS

Thanks to everyone.

REFERENCES

[1] Brewer, E. (2010, July). A certain freedom: thoughts on the CAP theorem. In *Proceedings of the 29th ACM SIGACT-SIGOPS symposium on Principles of distributed computing* (pp. 335-335).

[2] Gilbert, S., & Lynch, N. (2012). Perspectives on the CAP Theorem. *Computer*, *45*(2), 30-36.

[3] <https://jepsen.io/>

[4] Burckhardt, Sebastian. "Principles of eventual consistency." (2014).

[5] Emmi, Michael, and Constantin Enea. "Weak-consistency specification via visibility relaxation." *Proceedings of the ACM on Programming Languages* 3.POPL (2019): 1-28.

[6] Biswas, Ranadeep, and Constantin Enea. "On the complexity of checking transactional consistency." Proceedings of the ACM on Programming Languages 3.OOPSLA (2019): 1-28.

[7] Wang, Chao, et al. "Replication-aware linearizability." Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation. 2019.

[8] Emmi, Michael, and Constantin Enea. "Monitoring weak consistency." International Conference on Computer Aided Verification. Springer, Cham, 2018.

[9] <https://github.com/elem-azar-unis/CRDT-Redis>Conference Name:ACM Woodstock conference

Conference Short Name:WOODSTOCK’18

Conference Location:El Paso, Texas USA

ISBN:978-1-4503-0000-0/18/06

Year:2018

Date:June

Copyright Year:2018

Copyright Statement:rightsretained

DOI:10.1145/1234567890

RRH: F. Surname et al.

Price:$15.00