SyncFree

SyncFree Technology White Paper

Commander: Bug-Finding for Programs Running on Weakly Consistent Platforms

Maryam Dabaghchian
Koç University

1 August 2016

Introduction

Modern geo-replicated CRDT key-value data stores provide high availability by reducing the synchronization between replicas. They provide weaker consistency guarantees, called *eventual consistency* [3], that guarantees all updates performed at every replica will eventually be propagated and become visible on all remote replicas.

Applications layered over such CRDT key-value data stores expose a wider set of program behaviors through the reduced consistency guarantees of the data store. Due to the non-determinism between performing an update on a replica and delivering updates coming from other replicas, an operation may access stale data causing invariant violation. So, some possible execution order can violate the application invariants, even though the default execution runs satisfying the invariants. Thus, application programmers need to consider every possible ordering of concurrent updates while writing a program for a weakly consistent platform.

Systematic Exploration of Program Behaviors

To help programmers develop correct applications running on weakly consistent platforms, we propose a runtime verification approach that systematically explores possible behaviors of such applications to detect application invariant violations arising from operations reordering. Our approach explores different ordering of performing any update on a replica and delivering any update coming from other replicas. It allows only one replica to proceed at a time in order to enable accessing the whole system state after each update. Our systematic exploration algorithm is called *causality-aware delay-bounding* (CAD) which is a variant of delay-bounded scheduling technique [2] that is an effective program search space prioritization technique used for exploring different behaviors of concurrent programs. Also, CAD takes the causal consistency into consideration while exploring different possible schedules. That is, it only explores those schedules using delay-bounding that are allowed by causal consistency.

Commander: A runtime verification tool

We have implemented our runtime verification technique as a tool called *Commander* which utilizes CAD algorithm to explore possible program behaviors at runtime and check if specified program invariants by the programmer are satisfied. Commander has been implemented to check applications developed for Antidote data store which is a CRDT key-value transactional data store. In order to use Commander, the programmer writes a test case for her application using specified callback functions by the Commander. Application invariants are also specified using any arbitrary Erlang code through a callback function. Then, the Commander systematically generates and explores different possible schedules using the test case, and checks application invariants.

Commander first runs the test case, and records the ordering in which transactions are executed or delivered to remote replicas, as the original schedule. Then, it starts generating and exploring all possible schedules around the original schedule. We have instrumented Antidote, in order to record transaction information while running the test case, and also to replay a transaction execution or a transaction delivery to a remote replica while exploring all possible schedules.

How Commander Works

We propose CAD algorithm to prioritize the program search space to explore possible schedules of a program systematically. To generate a new schedule, CAD allows delaying bounded number of tasks, that in our setting are transaction executions or transaction deliveries to remote replicas. Scheduling a transaction execution may form a new causality dependency than the one the original schedule has, but then the formed causality dependency is preserved through the whole schedule while scheduling remaining tasks.

Commander consists of four components: *Recorder*, *Scheduler*, *Replayer* and *Verifier*. Commander first runs the test case written by the programmer, and the *Recorder* records the ordering of transactions execution or their delivery to remote replicas as the original schedule. Using the original schedule, *Scheduler* component generates other possible schedules using CAD algorithm. For each schedule, *Replayer* components replayes the next scheduled task

allowing only one replica to operate at a time. After replaying a task, *Replayer* calls *Verifier* component to check the application invariants specified by the programmer. If the application invariants are satisfied, *Verifier* acknowledges the *Replayer* to continue by replaying the following scheduled task. Otherwise, *Verifier* provides a counter example to the programmer.

Related Work

In order to ease programming applications layered over weakly consistent data stores, Balegas *et al.* [1] proposed a form of consistency called *explicit consistency* that strengthen the eventual consistency for unsafe operations under concurrent execution, to ensure the application invariants. To define a more stringent consistency model, applications define the consistency rules as invariants. Then, a *reservation* system is derived based on the application and its invariants. The derived reservation system is used to enforce the specified explicit consistency by preventing concurrent execution of unsafe operations. Escrow reservation, which is a reservation technique used for numeric invariants, allows only a limited number of an unsafe operation to execute without coordination. However, these approaches need the programmer to specify the consistency model required by the program as invariants. So, the correctness of the technique relies on the correctness of the consistency rules defined by the programmer.

Using Commander, the programmer need to specify only the application invariants, but not the consistency model. On the other hand, the programmer can strengthen the consistency model of the application by choosing an appropriate CRDT data type. For instance, pn_counter and bounded counter CRDTs provide different levels of consistency. Thus, depending on the program behavior, the programmer can use pn_counter or bounded counter data type to enforce the desired consistency guarantee.

To the best of our knowledge, our approach is the first runtime verification approach to check the correctness of distributed applications running on weakly consistent platforms.

References

[1] Valter Balegas, Nuno Preguiça, Rodrigo Rodrigues, Sérgio Duarte, Carla Ferreira, Mahsa Najafzadeh, and Marc Shapiro. Putting consistency back into eventual consistency. pages 6:1–6:16, Bordeaux, France, April 2015.

[2] Michael Emmi, Shaz Qadeer, and Zvonimir Rakamarić. Delay-bounded scheduling. In *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '11, pages 411–422, New York, NY, USA, 2011. ACM.

[3] D. B. Terry, M. M. Theimer, Karin Petersen, A. J. Demers, M. J. Spreitzer, and C. H. Hauser. Managing update conflicts in bayou, a weakly connected replicated storage system. In *Proceedings of the fifteenth ACM symposium on Operating systems principles*, SOSP '95, pages 172–182. ACM, 1995.