March 2023

Reducing Time to Reset a Stateful System Under Test (SUT)

Souvik Paul

Andrew Khatko

Frank Legler

Shubham Sharma

RK Palabindela

Reducing Time to Reset a Stateful System Under Test (SUT)

ABSTRACT

The qualification of continuously running, stateful systems requires a full reset to initial state due to the state-altering nature of a finite workload. A full reset entails teardown and restart of the system to a known initial state, which can cause the workload to experience long downtimes for the duration of the reset. This disclosure describes techniques that reduce reset durations that occur in the context of testing stateful systems. In a one-time operation, an initial state is maintained in persistent storage, e.g., in cloud-based virtual disks. The system under test (SUT) is reset using the stored initial image. The time saved in the obviated teardown-and-restart procedure directly translates to reduced startup and qualification times and obviates other time-consuming procedures such as reinstalling dependent libraries and software. The techniques generally reduce interruptions in testing and are especially useful for narrow rollout windows.

KEYWORDS

- System under test (SUT)
- Stateful systems
- Regression testing
- System qualification
- System testing
- Qualification workload
- Software rollout
- Software testing

BACKGROUND

Where software is built from multiple individual binaries, a continuously running workload, e.g., a combination of system under test (SUT), data, and simulated load, is used to qualify the system. An example SUT in the context of cloud computing is a virtual machine. Downtime in the qualification process of a SUT can result in defects leaking through a binary release.

The qualification of continuously running, stateful SUTs requires a full reset to initial state due to the state-altering nature of a finite workload. A full reset entails teardown and restart of the system to a known initial state, which causes the workload to experience a downtime for the duration of the reset. For complex systems, the reset duration can be long, e.g., of the order of tens of minutes to hours. Aside from the intrinsic undesirability of downtime, long downtimes allow the possibility of new regressions arising due to new rollouts. Unnoticed regressions (or delay in catching issues) can surface to end users as bugs.

DESCRIPTION

This disclosure describes techniques that reduce the reset duration that occurs in the context of testing stateful systems. The techniques generally minimize interruptions in testing and are especially useful for narrow rollout windows, e.g., rollouts that occur every few hours. Reset durations can be reduced from the current order of hours to orders of minutes. The reduced reset durations enable regressions to be identified before the rollout window and blocked from going forward. Continuously running workloads experience minimal interruption.

An initial state is maintained in persistent storage, e.g., in cloud-based virtual disks. The initial state can be saved in a one-time operation where the state image is pre-baked into the disk. In contrast to conventional teardown-and-restart of the SUT, resetting of the SUT is achieved using the pre-baked state image. The time saved in the obviated teardown-and-restart procedure directly translates to reduced startup and qualification times. Using a pre-baked state image, as described herein, also obviates other time-consuming procedures such as reinstalling dependent libraries and software.

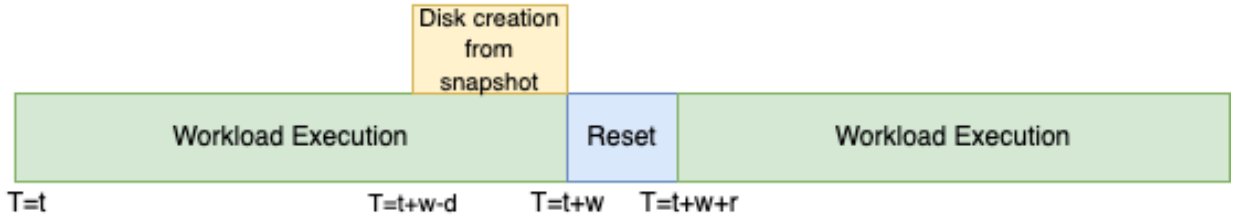

Fig. 1: Timeline of events leading to the reset and thereafter

Fig. 1 illustrates an example timeline of events leading to the reset and thereafter. A workload execution by the SUT on generated traffic lasts for *w* seconds, e.g., between times *t* and *t+w*. The time needed to create a disk from the snapshot stored in persistent storage is *d*. The reset duration is *r*. At a time *d* seconds prior to the completion of execution or earlier, disk creation is initiated. To minimize the reset duration, disk creation is done parallel to execution. Once created, the disk is used to reset the state of the SUT. Upon the completion of reset, execution resumes. The execute-reset-execute cycle is repeated.

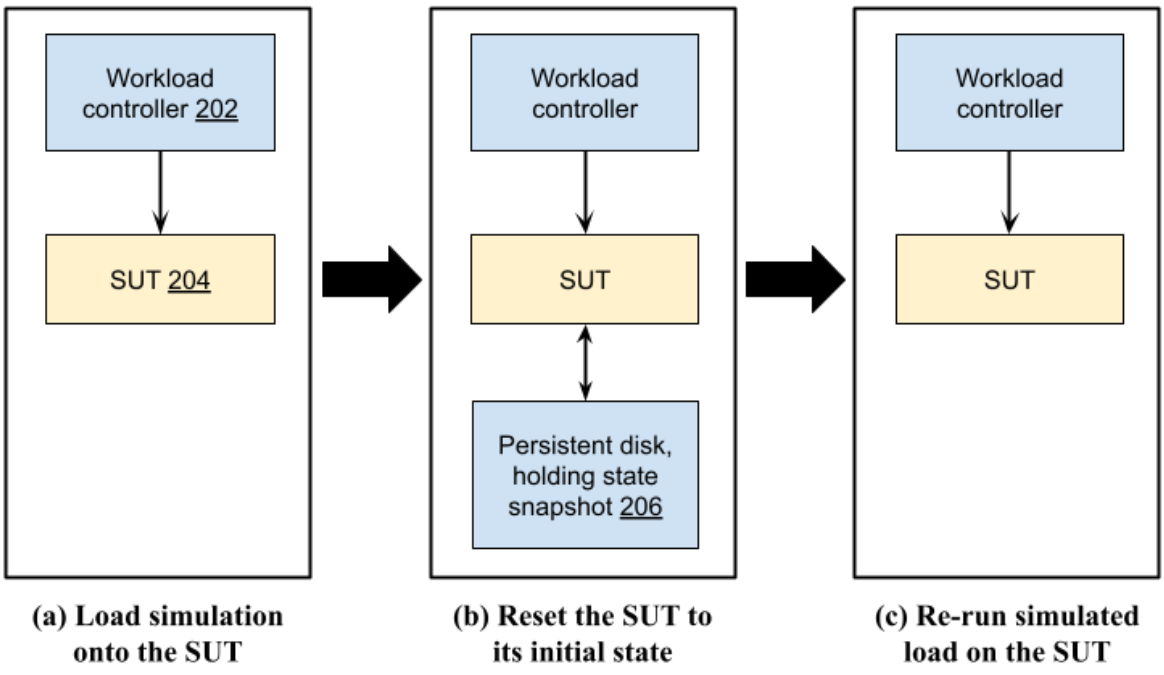

Fig. 2: Framework for controlling the system under test (SUT)

Fig. 2 illustrates an example framework for controlling the SUT. A workload controller job (202) running outside the workload loads the SUT (204), resets it, and otherwise interacts with the SUT as necessary. An initially generated load is executed by the SUT, as shown in Fig. 2(a). Once completed, the SUT is reset to its initial state, as shown in Fig. 2(b). The reset can be done by replacing the persistent disk of the SUT with a new disk created from the initial image of the persistent disk (206). Once reset, the load is re-executed by the SUT, as shown in Fig. 2(c).

The described techniques apply to any situation where a SUT has to be qualified under a continuously running qualification workload of fixed size. Alternative to the described techniques, a different instance of the system can be brought up in parallel to the one currently under test and the next cycle of traffic can be executed by the parallel SUT. This alternative approach is more costly, requires a larger maintenance overhead, and may require reinstallation of the dependencies of the SUT.

CONCLUSION

This disclosure describes techniques that reduce reset durations that occur in the context of testing stateful systems. In a one-time operation, an initial state is maintained in persistent storage, e.g., in cloud-based virtual disks. The system under test (SUT) is reset using the stored initial image. The time saved in the obviated teardown-and-restart procedure directly translates to reduced startup and qualification times and obviates other time-consuming procedures such as reinstalling dependent libraries and software. The techniques generally reduce interruptions in testing and are especially useful for narrow rollout windows.