Technical Disclosure Commons

Defensive Publications Series

April 11, 2018

Real-time scheduling for software testing

Keun Soo Yim

Yuexi Ma

Real-time scheduling for software testing

ABSTRACT

Software is subjected to a large variety of tests on different devices prior to public release, including pre-submit and post-submit tests, that have different scheduling attributes. The number of physical devices available for testing is limited, e.g., due to the devices being unreleased or under development. This disclosure provides techniques for scheduling software tests on a scarce pool of physical devices. The tests have differing scheduling attributes, e.g., durations, periodicity, degree of preemptiveness, etc. Tests are scheduled such that a test experiences low latency (queueing delay) regardless of test duration, the availability of the device pool is improved, and device underutilization/ instances of overloading are reduced. The techniques improve test scheduling and can enable improved engineering and application-developer productivity and can help reduce time to market for new devices.

KEYWORDS

- pre-submit test
- post-submit test
- time to market
- device testing
- software test

BACKGROUND

Software such as mobile application, operating system, etc., is subject to extensive testing prior to public release. In a typical scenario, a test laboratory utilizes a pool of devices, e.g., unreleased or under development devices such as smartphones. The devices are provided for testing by various manufacturers. Software developers download test versions of their

software to devices in the test laboratory. Alternatively, the test laboratory monitors a code-base for new test versions of code and performs test of such software using lab devices. Further, some software developers have a pool of devices attached to their workstations, and test new code on the local pool of devices.

For efficient software testing and development, some short tests are run prior to check-in (submission) of code to a code-base. These tests, referred to as pre-submit tests, typically complete quickly, e.g., within minutes. Other tests are run when the code is at a greater state of maturity, e.g., after check-in or submission of code. These tests are referred to as post-submit tests, and can take a longer duration to execute, e.g., forty hours for a compatibility test suite. Still other types of tests use multiple devices simultaneously, e.g., to test peer-to-peer communication features.

Given the numbers of software applications and versions, scale of code development, and count of new devices entering the marketplace, there is a near-perennial shortage of devices on which a given version of software can be tested. A lengthy post-submit test can cause a long queue in applications that await testing, simply due to the length of time required to complete the test, Similarly, a multi-device test causes excess load on a scarce pool of devices available for testing. Tests that require a specific set of devices may not be able to execute due to a temporary unavailability of such devices.

One solution to the relatively large spread in duration or scheduling attributes of tests is to split the pool of available devices into a pre-submit pool, e.g., for relatively short tests with unpredictable launch times, and a post-submit pool, e.g., for relatively long tests with periodic launch times. The pre-submit pool runs test modules with many smaller test cases. The post-submit pool runs long-duration single-test cases. This permits tests that require short testing

durations to have a dedicated pool of devices such that these tests do not get gated by tests that require long testing durations. It also prevents interruption of long-duration single-test cases. However, such a splitting of the pool of devices sometimes results in devices that go unutilized. For example, a device in the pre-submit pool may be free, but is still unavailable for execution of a post-submit test. Also, a split in the pool of devices, as described above, does not efficiently handle the observed spikes or lulls in pre-submit test requests that occur at certain times of the day. Thus, the pool of devices is often either underutilized, or conversely, a large queue of test requests accumulate, awaiting devices to be allocated for tests.

DESCRIPTION

This disclosure describes techniques to address the problem of underutilization or overloading by unifying the pre-submit (e.g., short-duration) and post-submit (e.g., long-duration) pools of devices to create a single pool of devices that can be used for either short-duration or long-duration tests. In this manner, a larger number of devices becomes available for pre-submit testing at times of high demand.

In the event that post-submit (long-duration) tests clog the pool of devices amid a growing number of requests for pre-submit (short-duration) tests, one or more long-duration tests is suspended or paused, with intermediate results saved. Short-duration tests are launched on these just-released devices. Once short-duration tests are disposed, relatively quickly, e.g., within tens of minutes to one hour, previously-stored intermediate results of the long-duration tests are re-loaded and the tests resume from the point of suspension.

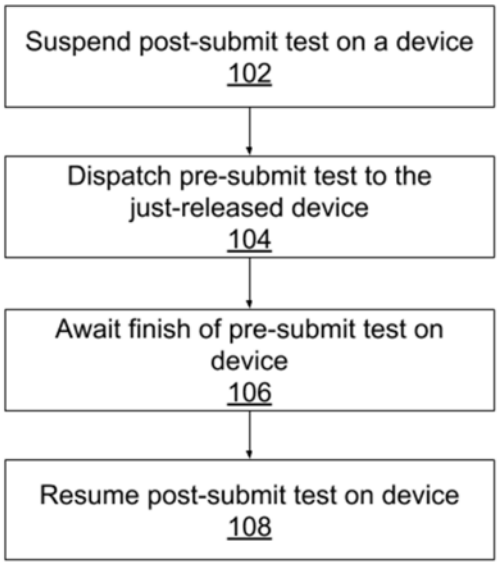

Fig. 1: Scheduling pre-submit and post-submit tests on a pool of devices

Fig. 1 illustrates the steps involved in scheduling pre-submit (e.g., relatively short) tests on a device pool that is occupied with post-submit (e.g., relatively long) tests. A running post-submit test is suspended and the current state of the post-submit test, including results of all thus-far executed test modules, is saved (102). A test generally includes a hierarchy of sections, e.g., test-plan, test-module, test-case, with test-case representing the smallest granularity of test-scheduling unit.

Conceptually, a command to suspend a running post-submit test may act in one of several ways, as follows.

- precise suspension, wherein the currently running test case or module is allowed to execute to completion prior to suspension of the post-submit test;
- in-precise suspension, wherein the currently running test case or module is immediately stopped, and when re-started, resumes from the last finished test case or module;

- checkpoint suspension, wherein the currently running test case or module is checkpointed, and when re-started, resumes from the checkpoint, which is within the currently running test case or module; etc.

A command to suspend a running post-submit test ideally travels through and is effective throughout the entire hierarchy of test sections. Further, to forestall device instability or lock-up in unknown state, the presently running test-case, which is the smallest granularity of test-scheduling unit, is allowed to execute to completion prior to suspension of the post-submit test (precise suspension). A test-case has three phases, e.g., startup, execution and clean-up. By allowing the test-case to complete, the clean-up stage is exercised, thereby assuring that the device remains in a known, stable state after test suspension.

Once the device is available, pre-submit test(s) are dispatched to the just-released device (104). The release of devices facilitates multi-device testing. For example, if there are two devices, one idle and the other used for post-submit testing, then upon release of the second device, both devices are used for a two-device pre-submit test.

The pre-submit tests are run to completion (106). As mentioned above, pre-submit tests generally complete faster, e.g., within minutes-to-an-hour, than post-submit tests, which may take tens of hours to complete. The post-submit test is resumed on the device (108). As with test suspension, test resumption is achieved throughout the hierarchy of test sections, e.g., test-plan, test-module, test-case. In addition, test resumption can include retrieval and re-execution of previously failed test cases. Further, the post-submit test can resume either on the device that just finished pre-submit testing, or on another equivalent device.

The techniques of this disclosure apply more generally to the problem of real-time scheduling of tasks that differ in length, periodicity, and/or preemptiveness. In the testing

scenario described herein, post-submit tasks are relatively long, predictable in occurrence (e.g., periodic, with period once per day or week), and preemptive. Pre-submit tasks are relatively short, generally unpredictable in occurrence or aperiodic, and non-preemptive. The techniques of this disclosure enable improved scheduling for a mix of tasks with heterogeneous scheduling attributes.

By using the same pool of devices for pre-submit and post-submit testing, device availability is improved. Device utilization is also improved, even as short-duration (pre-submit) tests are not made to suffer unduly long queueing delays. In this manner, a productivity gain is achieved for the testing process.

CONCLUSION

This disclosure provides techniques for scheduling software tests on a scarce pool of physical devices. The tests have differing scheduling attributes, e.g., durations, periodicity, degree of preemptiveness, etc. Tests are scheduled such that a test experiences low latency (queueing delay) regardless of test duration, the availability of the device pool is improved, and device underutilization/ instances of overloading are reduced. The techniques improve test scheduling and can enable improved engineering and application-developer productivity and can help reduce time to market for new devices.