September 20, 2018

Distributed and Collaborative Test Scheduling to Determine a Green Build

Keun Soo Yim

Distributed and collaborative test scheduling to determine a green build

ABSTRACT

In the parlance of software testing and verification, a green build is a software build that passes tests on all reference devices. A green build is typically determined by a centralized test-scheduler. The centralized test-scheduler has a database of parameters, e.g., build-artifacts, build-branches, etc., corresponding to each device. The centralized scheduler uses the database to efficiently schedule tests. Centralized scheduling is computationally intensive, and maintenance of the database is a significant burden.

Per the techniques of this disclosure, devices in a pool collaboratively pick a new build to test. The first device to start within a given scheduling interval picks a build, and the remaining devices pick the same build. The devices independently test the selected build. The first device to finish testing, either due to pass or fail, picks another build. The remaining devices follow the newly picked build. The process continues until the devices converge upon a green build. The distributed manner of test scheduling, as described herein, enables efficient determination of the green build.

KEYWORDS

Test scheduler; green build; build-target; build-artifact; test-package; build-branch; primary branch; manifest branch; decentralized scheduling; distributed scheduling; collaborative scheduling; affinity group; tip-of-tree

BACKGROUND

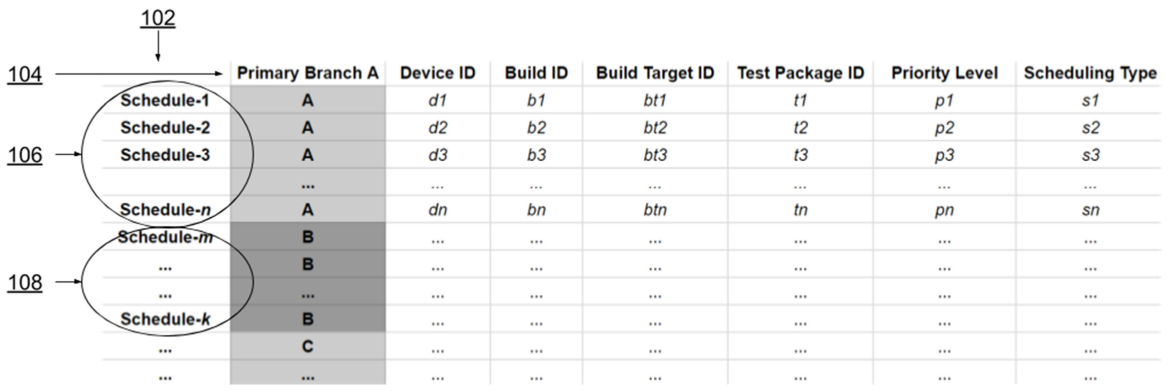
A software product, e.g., operating system, platform, application, etc., is typically designed to work on multiple device types. The software is therefore tested across multiple target devices prior to release. A green build is a software build that passes tests on all reference devices. A recent green build is used for product release.

A green build is typically determined by a centralized test-scheduler. The centralized test-scheduler has a database of parameters, e.g., build-artifacts (and combinations thereof), build-branches, etc., corresponding to each device. The centralized scheduler uses the database to efficiently schedule tests. Centralized scheduling is computationally intensive. Maintenance of the database is a significant burden, e.g., as devices enter and leave the test pool.

DESCRIPTION

Per the techniques of this disclosure, devices in a pool collaboratively converge to a new green build in a decentralized manner. This disclosure uses the following definitions in this context:

- Test schedule: A test schedule is a record or a tuple, e.g., as shown in Fig. 1, comprising test parameters such as primary branch of code-under-test, device ID, build ID, build-target ID, test-package ID, priority level, scheduling type, etc.


102
104
106
108

	Primary Branch A	Device ID	Build ID	Build Target ID	Test Package ID	Priority Level	Scheduling Type
Schedule-1	A	*d1*	*b1*	*bt1*	*t1*	*p1*	*s1*
Schedule-2	A	*d2*	*b2*	*bt2*	*t2*	*p2*	*s2*
Schedule-3	A	*d3*	*b3*	*bt3*	*t3*	*p3*	*s3*
	...	...	...	...	...	...	...
Schedule-*n*	A	*dn*	*bn*	*btn*	*tn*	*pn*	*sn*
Schedule-*m*	B	...	...	...	...	...	...
...	B	...	...	...	...	...	...
...	...	...	...	...	...	...	...
Schedule-*k*	B	...	...	...	...	...	...
...	C	...	...	...	...	...	...
...	...	...	...	...	...	...	...

Fig. 1: A set of test schedules

The example of Fig. 1 illustrates a number of test schedules (102), each comprising test parameters (104). Other test parameters can be included within a test schedule, e.g., scheduling frequency, extra/special-purpose hardware needed by the device, hardware configuration and requirements, peripherals, etc. If devices are clustered logically, then cluster-ID is typically included within a test schedule.

- Primary branch or manifest branch: Primary branch is a branch of the code-under-test that is substantially independent of hardware. The primary branch is common and largely invariant across several devices or device types. The primary branch is usually specific to an architecture, e.g., RISC, x86, x64, etc. A primary branch can produce a build-type, e.g., a specific image characterized by build-artifacts. A complete platform, e.g., version of operating system, uses a manifest branch. A manifest branch can have multiple build targets, each of which can define artifact type. Thus, within a manifest branch there could be multiple builds.
- Affinity group: Affinity group is a set of test schedules that share the same primary branch (or branches, if multiple builds are used). In the example of Fig. 1, schedules *1* through *n* form an affinity group (106), since these schedules share the same primary branch (A). Schedules *m* through *k* form another affinity group (108), sharing as they do a primary branch (B). Per the techniques of this disclosure, the affinity group forms a basic unit of scheduling. An affinity group can include any number of devices, e.g., between two and hundreds of devices.

To find a green build efficiently, the techniques of this disclosure run one of two procedures, referred to as baseline procedure and extended procedure, every scheduling period, e.g., every twenty-four hours.

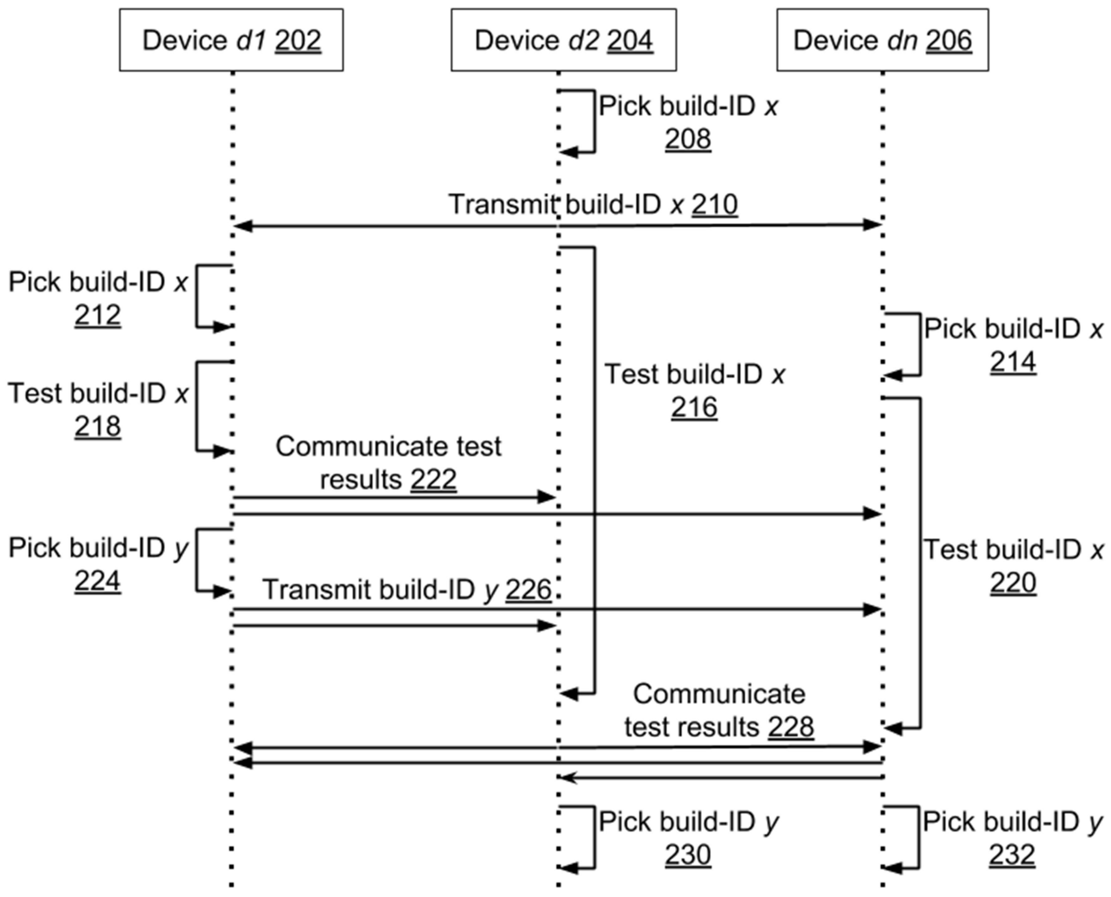


Fig. 2: Baseline procedure to find green build

Baseline procedure

The baseline procedure is illustrated in Fig. 2. A number of devices (202-206) belong to a single affinity group. At the start of the scheduling period, a first device picks a build-ID *x* (208). The build-ID can be picked in a number of ways, e.g., as any build-ID later than the previously known green build, or as the tip-of-tree (latest) build-ID, or in a manner optimal to the device (204) that is first picking a build-ID.

The device that first picks a build-ID transmits (210) that build-ID to other devices within the affinity group. Other devices in the affinity group pick the same build-ID *x* (212, 214). Each device tests the build independently (216-220). The device that finishes testing first

(218), e.g., by passing or by failing, communicates test results (222) to other devices in the affinity group, and picks the next build-ID *y* (224). The device transmits the newly picked build-ID *y* (226) to other devices of the affinity group, and starts testing the build-ID *y*.

Whenever a next device finishes its tests, e.g., by passing or by failing, it communicates test results to other devices in the affinity group (228), and picks for its next test the same build-ID *y* (230, 232) that was picked by the first device that finished the first test run. This procedure continues until a green build, e.g., a build that passes in all devices of an affinity group, is found. In this manner, test scheduling is decentralized to the devices within an affinity group and coordinated between these devices. The picking of a particular build by a given device is left to the device, so long as it is more recent than the previously known green build, if any. The picking of a new build by a device is decoupled from the coordination process.

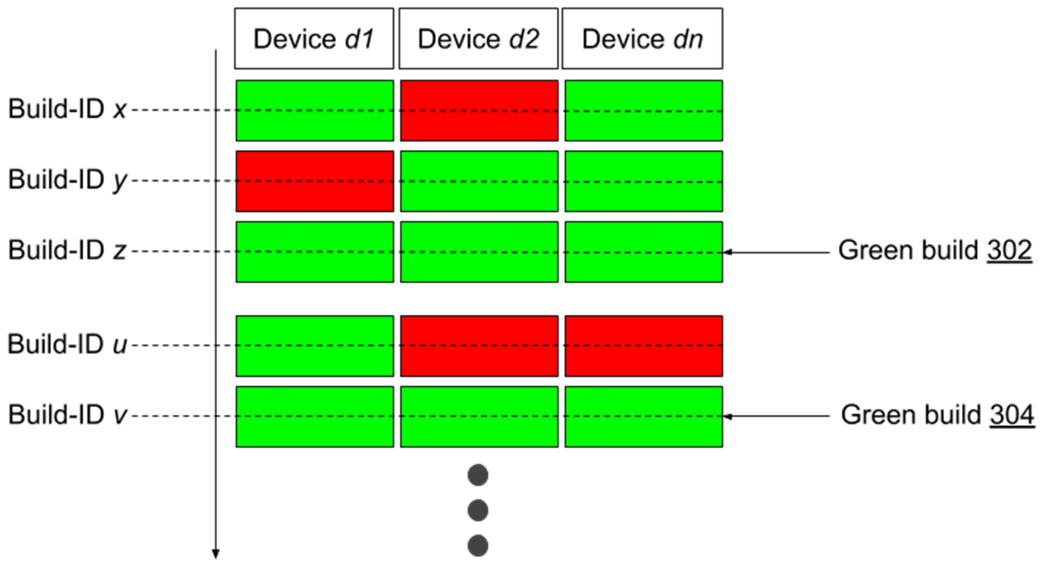


Fig. 3: Green build determination with the passage of time

Fig. 3 illustrates the determination of green builds as time passes. Builds *x*, *y*, and *z* are each builds more recent than the previously known green build, if any. Builds *x*, *y* and *z* are sequentially selected per the baseline procedure. At build *x*, a test at device *d2* fails (indicated by red), and at build *y*, a test device *d1* fails. At build *z*, tests at all devices pass (indicated by green) and build *z* becomes the new green build (302).

The baseline procedure continues, with new builds *u* and *v* selected sequentially, where both build *u* and *v* are more recent than the latest green build (*z*). Build *u* fails a test on at least one device, but build *v* passes tests on all devices and becomes the new green build (304).

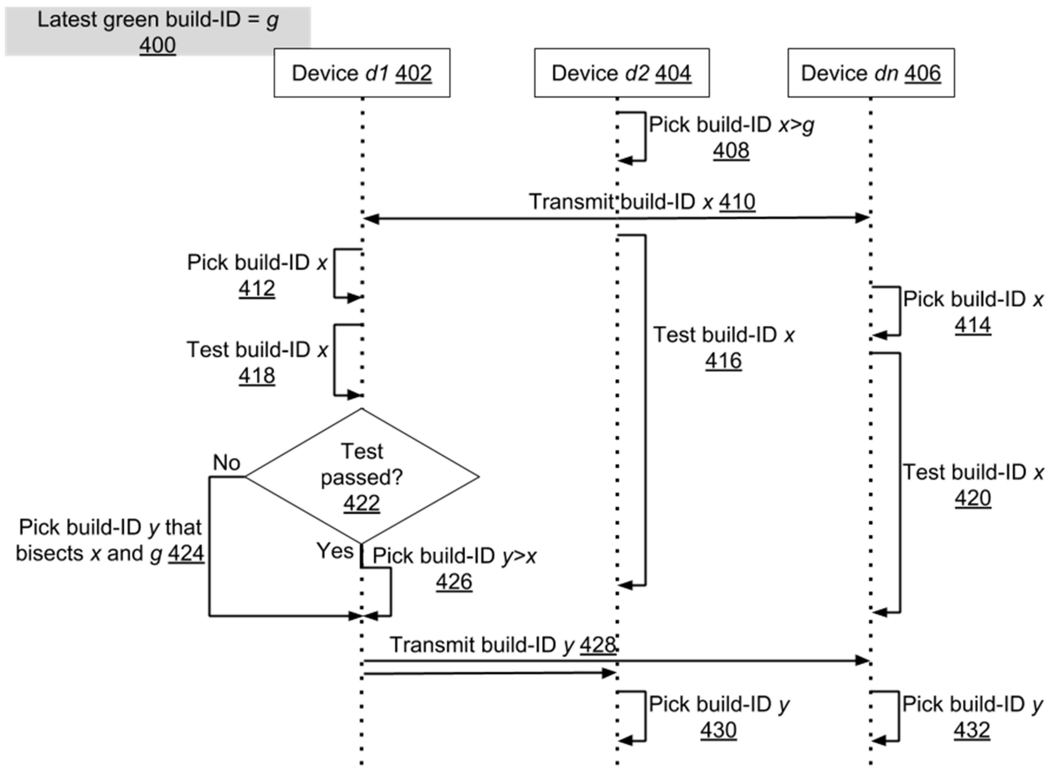


Fig. 4: Extended procedure to find green build

Extended procedure

The extended procedure is illustrated in Fig. 4. A number of devices (402-406) belong to a single affinity group. At the start of the scheduling period, the most-recent green build is *g* (400). A first device picks (408) a build-ID *x* that is later than the most-recent green build, and transmits (410) that build-ID to other devices in the affinity group. Other devices pick (412, 414) the same build-ID *x*.

Each device starts testing build-ID *x* (416-420) independently. The first device that finishes testing build-ID *x* assesses the test result (422). If the test passes, then it picks a new build-ID *y* that is more recent than *x* (426). If the test fails, then it picks a build-ID *y* that is between, or bisects, the last-known green build *g* and the most recently failed build *x* (424). The build-id *y* is transmitted to other devices in the affinity group (428), and each device picks build-ID *y* (430-432). Testing proceeds on build-ID *y*, and the process repeats until and beyond the point a new green build is found. In this manner, the extended procedure allows for both forward and backward movement in build-ID in order to determine latest green build.

A number of variations are possible on the extended procedure as described above. For example, upon the first failure of a build within an affinity group, the event of failure may be communicated to other devices in the affinity group, and the other devices may stop testing that build. The next selection of build-to-be-tested, e.g., one that is between or bisects *g* and *x,* may be done by another device in the affinity group.

As another example variation, after a certain number of unsuccessful bisections, a build may be selected as between the last-known failed build and the tip-of-tree (last known build), thus allowing for a forward movement in build-ID after a certain number failed backward movements. As a further example variation, if more than a certain number of devices failed, the tip-of-tree is selected with a pre-determined probability.

A test suite comprises several test modules, which in turn comprises several test cases. While a green build has the entire test suite passing, test failures may happen in only certain modules or suites. When selecting a new build after a failure, e.g., by bisecting the last-known green build and the most-recent failure, only failed modules or cases may be tested at first or at all. In this manner, the techniques of this disclosure find a new green build efficiently without centralized coordination.

CONCLUSION

A green build, which is a software version that passes on all reference devices, is typically determined by a centralized test-scheduler. Centralized scheduling is computationally intensive. Per the techniques of this disclosure, devices in a test pool collaboratively pick a new build to test. The first device to start within a given scheduling interval picks a build, and the remaining devices pick the same build. The devices independently test that build. The first device to finish testing, either due to pass or fail, picks another build. The remaining devices follow the newly picked build. The process continues until the devices converge upon a green build. The distributed manner of test scheduling, as described herein, enables computationally efficient determination of the green build.