July 24, 2019

Virtual devices as a service

Tracy Wang

Rim Yazigi

Virtual devices as a service

ABSTRACT

Software applications are developed and tested over a large and evolving variety of devices of different device types. Development and testing with physical devices is tedious and time consuming and has scaling and reliability problems. Per techniques of this disclosure, a large pool of virtual devices is instantiated on a compute cluster and made available to software developers as a service. Developers check out as many virtual devices as needed, conduct test and development activity, reset the devices, and release the devices back to the pool. The techniques obviate the need for physical devices and the concomitant issues of cost and reliability and enable large scale testing and development and faster device releases.

KEYWORDS

- Software testing
- Device pool
- Virtual device
- Device farm

BACKGROUND

Software applications are developed and tested over a large and evolving variety of devices, device types, and software platforms. For example, applications are designed to function and interoperate across devices such as smartphones, tablets, smart speakers, personal computers, smart home devices (smart lights, smart hubs, smart entertainment devices, etc.), devices that cast to screens or speakers, etc.

Development and testing with physical devices is tedious and time consuming and has scaling and reliability problems. Maintaining a farm of different types of physical devices for

server-side testing is resource intensive. Manual set up and pairing or wiring of physical devices, especially running alpha or beta software, is error-prone. The test environment itself can be subject to WiFi interference and stability issues.

DESCRIPTION

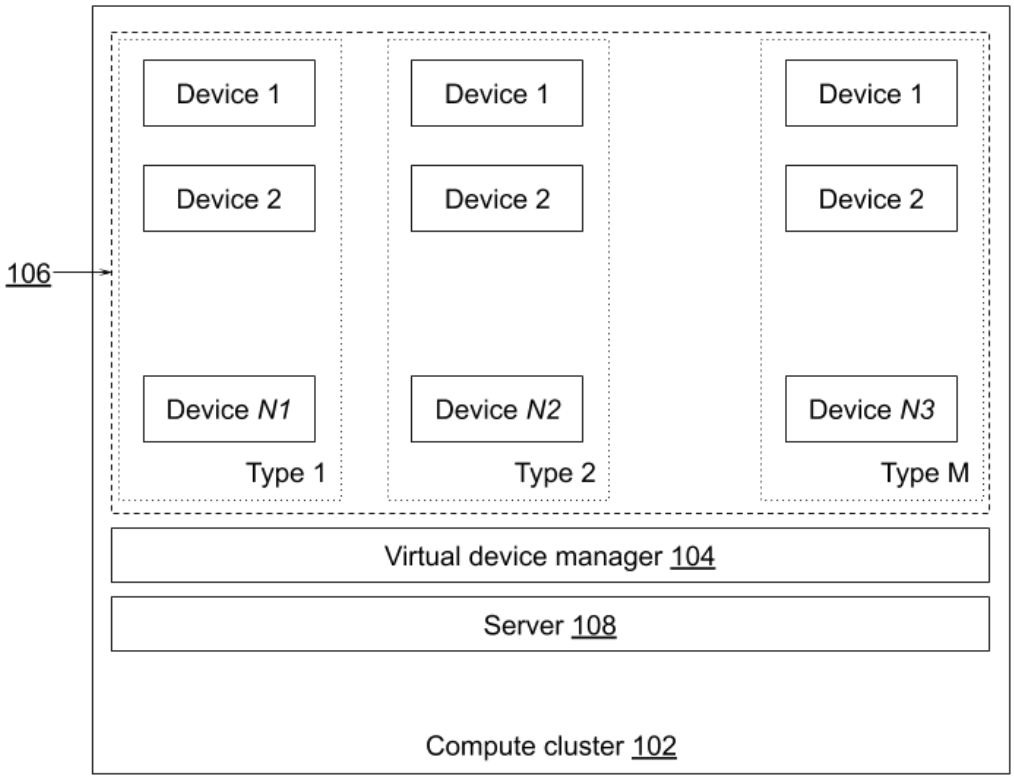

Fig. 1: Virtual test environment

As illustrated in Fig. 1, per the techniques of this disclosure, a large pool of virtual devices (106) of various types is instantiated on a compute cluster (102) that includes a server (108). The pool of virtual devices is managed by a virtual device manager (104). Multiple compute clusters are provided for server-device fault tolerance and interoperability. A load balancer is provided that distributes the load over the compute clusters. The pool of virtual devices is made available to developers as a service. The virtual devices emulate devices from a variety of manufacturers and operating systems, including the entire platform stack, and provide

a stable, scalable development environment. The virtual devices are configured with the same operating system and software as does the corresponding physical device, including the same software updates.

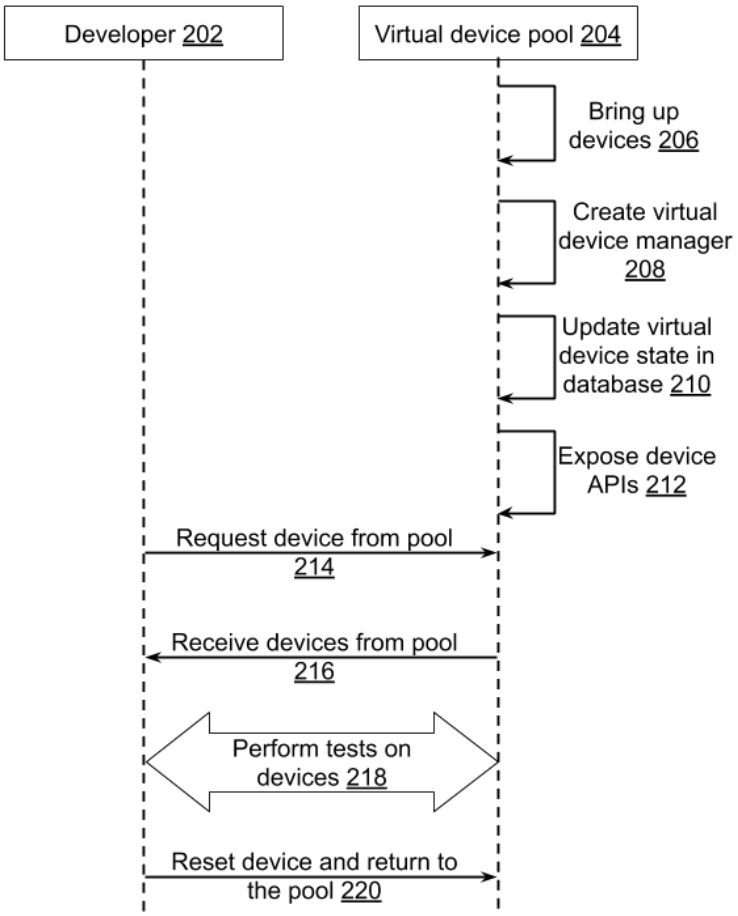

Fig. 2: Checking out and returning virtual devices to the pool

Fig. 2 illustrates an example process of checking out and returning virtual devices to the pool. A virtual device pool (204) is populated with virtual devices by bringing them up (206) in a compute cluster. The number and type of virtual devices is a configurable parameter. A virtual device manager (208) is created that manages the virtual devices, e.g., checks the status of each virtual device, updates in a central database the devices' states (210), e.g., whether the device is in use, ports, ID, status, etc. APIs are provided (212) that a developer can use to control the

virtual devices using actions such as issuing text queries, sending audio files, transmitting and receiving text-to-speech responses, capturing screenshots, checking DOM elements etc. In this manner, a pool of warmed-up virtual devices that are ready to run without any initialization delay is available as a service.

A developer (202) sends a request for one or more virtual devices of one or more types from the pool (214) and receives a grant of an appropriate number of virtual devices of appropriate types to the developer. The virtual devices that are provided to the developer (216) include test accounts that are automatically linked. The developer conducts various test and development activities (218) on the virtual devices, e.g., send queries ("play some music", "translate to French", etc.) and receive responses; verify screenshots and device state; test for client-server compatibility; test for inter-device interoperability; deploy and test new features; etc. and otherwise interact with the virtual devices. Upon completion of the test, the developer resets the virtual devices to the available state and releases the devices back to the pool (220).

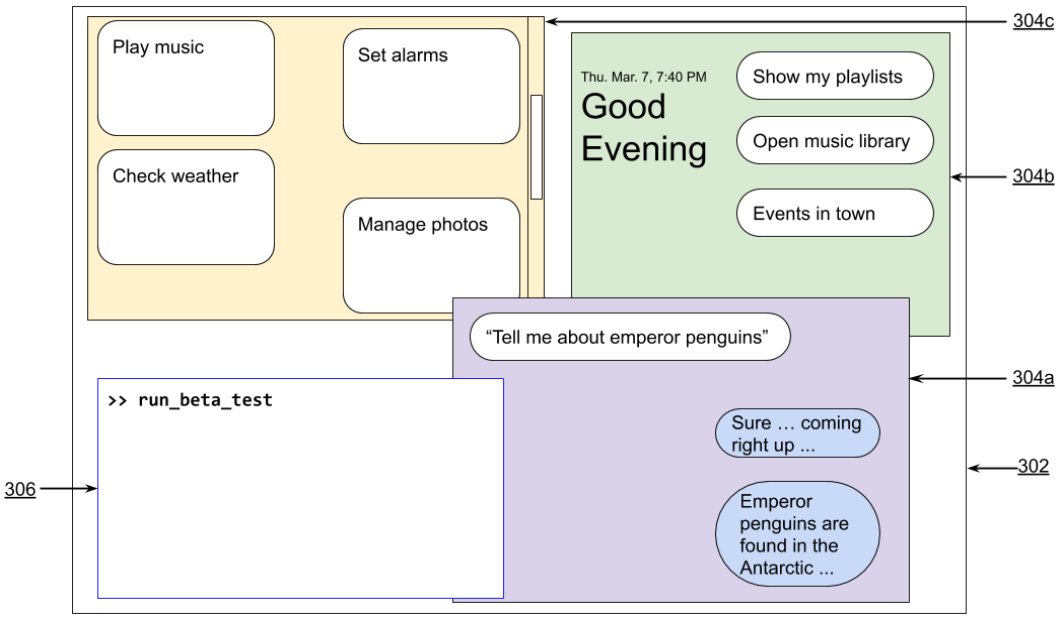

Fig. 3: A user interface to access virtual devices

Fig. 3 illustrates an example user-interface to access screen-based virtual devices. A developer can access via a physical personal computer, laptop, etc. (302) a number of virtual devices (304a-c), whose screens are displayed and can be interacted with via command line interface (306), via mouse clicks that are registered as touch events, etc. This enables testing of visual apps with platform software on virtual devices, without dependency on physical devices.

The virtual devices are derived from the same code base as the physical devices, such that testing on a virtual device bears high fidelity to the performance of a corresponding physical device. The virtual devices can simulate a client of a specific firmware version, enabling regression testing of particular versions.

Compared to a physical device farm, even one with a physical device manager, a virtual device farm is easy to maintain and scale. It is relatively easy to deploy several thousands of virtual devices, and run a very large number of tests or commands in parallel — something that would be much more expensive, less stable, require much supporting lab and test-bed infrastructure, and generally be much harder to do with physical devices, if at all. Virtual devices can be provided as a service to first-party, e.g., platform or operating system, developers, or, with authorization entry points, to third-party, e.g., application developers. As described herein, the burden of launching, maintaining, and resetting virtual devices is taken off software developers, allowing developers to focus on their task of developing reliable software.

Alternatively, an html representation of the visual response, including HTML rendering of cards/actions, can be hand-crafted or built to mimic real device appearance. While such a simulated client can work for UI displays, it lacks platform code and is not a real emulator that can support client operations.

CONCLUSION

Per techniques of this disclosure, a large pool of virtual devices is instantiated on a compute cluster and made available to software developers as a service. Developers check out as many virtual devices as needed, conduct test and development activity, reset the devices, and release the devices back to the pool. The techniques obviate the need for physical devices and the concomitant issues of cost and reliability and enable large scale testing and development and faster device releases.