The Journal of Systems and Software 124 (2017) 39–55

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

On building a cloud-based mobile testing infrastructure service system

Chuanqi Tao[a,b,*], Jerry Gao[c,d]

[a] Computer Science and Engineering Department, Nanjing University of Science and Technology, PR China
[b] State Key Laboratory for Novel Software Technology, Nanjing University, PR China
[c] Computer Engineering Department, San Jose State University, USA
[d] Taiyuan University of Technology, Taiyuan, PR China

ARTICLE INFO

Article history:
Received 16 April 2016
Revised 22 August 2016
Accepted 8 November 2016
Available online 9 November 2016

Keywords:
Mobile testing as a service
Cloud-based infrastructure -as-a-service
Mobile application testing

ABSTRACT

With the rapid advance of mobile computing, cloud computing and wireless network, there is a significant increasing number of mobile subscriptions. This brings new business requirements and demands in mobile testing service, and causes new issues and challenges. In this paper, informative discussions about cloud-based mobile testing-as-a-service (mobile TaaS) are offered, including the essential concepts, focuses, test process, and the expected testing infrastructures. To address the need of infrastructure level service for mobile TaaS, this paper presents a developed system known as MTaaS to provide an infrastructure-as-a-service (IaaS) for mobile testing, in order to indicate the feasibility and effectiveness of cloud-based mobile testing service. In addition, the paper presents a comparison among cloud-based mobile TaaS approaches and several best practices in industry are discussed. Finally, the primary issues, challenges, and needs existed in current mobile TaaS are analyzed.

Introduction

With the rapid advance of mobile computing technology and wireless networking, there is a significant increase of mobile subscriptions. This brings new business requirements and demands in mobile software testing, and causes new issues and challenges. The growing mobile market needs more and better testing approaches for mobile apps. A report from ABI Research predicted the growth of test automation will push the revenues close to $800 million by the end of 2017 (https://www.abiresearch.com). Mobile app vendors have encountered the following critical issues. Testing mobile apps and web applications on different mobile platforms and browsers on various devices becomes very costly and tedious due to the fast upgrading of mobile devices, large-scale mobile use access, rapid updates of mobile platforms and technologies, and fast upgrading mobile application services (Ridene and Barbier, 2011; Anand et al., 2012; Amalfitano et al., 2012; Satoh, 2004; Bo et al., 2007; Hargassner et al., 2008; Satoh, 2012; Muccini et al., 2012, Gao et al., 2013). According to Yang et al. (2010), Riungu et al. (2010), Ridene and Barbier (2011), Gao et al. (2011), Gao et al. (2014), Tao and Gao (2014), Aktouf et al., 2015, Buyya et al. (2009), Tsai et al., 2011, Anand et al. (2012), Amalfitano et al. (2012), Satoh (2004), Bo et al. (2007), Hargassner et al. (2008), Satoh (2012), Muccini et al., 2012, Gao et al. (2013) Gao et al. (2012), Bai et al. (2013), *testing-as-a-service* (TaaS) in a cloud infrastructure is considered as a new business and service model. A TaaS provider undertakes software testing project activities and tasks for under-test web-based software (or an application system) in a cloud infrastructure, and delivers them as a service for customers.

Cloud-based mobile TaaS offers a new business and service model for diverse mobile software validation services using the pay-as-you-test model to achieve cost-sharing and cost-reduction in mobile computing resources, networks, cloud computing and storage infrastructures. Therefore, cloud-based mobile TaaS is needed to resolve major issues in mobile application testing. These issues include: a) high costs in current mobile testing practice and environments; b) lack of testing support and tools for mobile scalability test; c) high mobile testing complexity and harness due to high diversity in mobile devices, platforms, browsers, and environments.

Although there are a number of published papers addressing issues, challenges and needs in mobile testing (services) (Gao et al., 2011; Gao et al., 2014; Tao and Gao, 2014; Aktouf et al., 2015; Buyya et al., 2009; Tsai et al., 2011; Anand et al., 2012; Amalfitano et al., 2012; Satoh, 2004; Bo et al., 2007; Hargassner et al., 2008; Satoh, 2012; Muccini et al., 2012; Gao et al., 2013; Gao et al., 2012), seldom publications discuss the challenges and needs in cloud-based mobile TaaS, especially at infrastructure level. In previous work (Gao et al., 2013; Gao et al., 2012), we mentioned some testing issues such as testing models, testing automation

* Correspondence author.
E-mail addresses: taochuanqi@njust.edu.cn (C. Tao), jerry.gao@sjsu.edu (J. Gao).

http://dx.doi.org/10.1016/j.jss.2016.11.016
0164-1212/

approaches in mobile testing and mobile TaaS. However, there are more challenges existed in testing criteria and standards, large-scale test automation and mobile TaaS. There are a lack of well-defined infrastructures and approaches which allow both mobile application vendors and users to access mobile TaaS services. In order to accommodate this, a suitable testing infrastructure is required which allows end users to submit on-demand service requests for mobile TaaS resources in order to form an infrastructure for mobile testing. This will include a large number of mobile devices, mobile emulators, mobile hubs, and computing server machines. Generally, an ideal mobile TaaS infrastructure must provide auto-provision, diversity support, and elastic scalability.

This paper focuses on mobile IaaS. To address the need for infrastructure service for mobile TaaS, an IaaS system MTaaS is developed to support the feasibility of mobile TaaS in practice. In addition, this paper provides informative perspectives on cloud-based mobile TaaS from service providers including the new features, current approaches and infrastructures, as well as issues, challenges, and needs.

The paper is structured as follows. The essential concepts, new requirements, and test process are presented in the next section. The testing infrastructures are introduced, and the current industry practices are compared in Section 3. A developed system for cloud-based mobile IaaS is presented in Section 4. The primary issues, challenges, and needs are discussed in Section 5. Conclusions are summarized in the end.

Understanding cloud based mobile testing-as-a -service

The term of "mobile testing" has been used to refer to different types of testing for mobile application testing, mobile device testing, wireless-based application testing, and mobile APP testing. In previous work, we initially defined mobile testing-as-a-service (Mobile TaaS) in cloud as follows (Gao et al., 2014).

"Mobile Testing as a Service (known as Mobile TaaS) provides on-demand testing services for mobile applications and/or SaaS to support software validation and quality engineering processes by leveraging a cloud-based scalable mobile testing environment to assure predefined given QoS requirements and service-level-agreements (SLAs)".

Here, mobile TaaS refers to a new testing service model and business model, as well as cloud-based testing infrastructure and platforms by leveraging cloud computing resources. Though some conventional testing features can be carried on in cloud-based mobile TaaS, most of them need to be addressed in cloud-based background due to the new features in cloud such as large-scale and on-demand.

2.1. Motivations and new requirements

Why do we need cloud-based mobile TaaS? The fast growing cloud-based mobile applications and the popularity of mobile cloud computing bring new needs and motivations. In previous work, we identified five primary reasons of mobile TaaS (Gao et al., 2014). They include: a) *high costs on mobile test infrastructures; b) frequent changes and upgrades on test platforms and devices; c) complex mobile user interfaces; d) large-scale on mobile test service, test simulation, and virtualization*; e) *multi-tenancy of mobile applications*. In addition, cloud-based mobile TaaS has four new requirements from the perspective of service as follows.

- It requires mobile *infrastructure-as-a-service* (IaaS) on cloud. It enables on-demand service for diverse mobile clouds, emulation service, test connectivity, TaaS servers mobile devices, hubs, etc.
- It requires mobile *platform-as-a-service* (PaaS) on cloud for users to perform mobile testing in terms of their needs, such as self-defined testing platform, test environment configuration, test script automation running, etc.
- It requires mobile testing service tools as *software-as-a-service* (SaaS) on cloud. Those SaaS applications can support cloud-based mobile testing at anytime and anywhere.
- It requires test service management, such as mobile resources manager, mobile environment manager, mobile test automation manager, mobile emulation manager, etc.

2.2. The test features of cloud-based mobile TaaS

According to our research survey and practical experience, cloud-based mobile TaaS primarily includes eight types of testing features. Five of the features are specified for cloud-based mobile TaaS. Three of them are focused in both general mobile testing and cloud-based testing. The features are listed as follows.

Elastic scalable mobile test infrastructure – This refers to the study of solutions on how to build elastic and scalable cloud-based infrastructure supporting automatic provision on mobile computing resources, such as mobile devices, emulators, platforms and browsers.

Mobile resource sharing, crossing platforms, and seamless accesses – This refers to test platform that enables to construct and set up a mobile test environment to meet diverse needs and requirements on mobile devices, including mobile platforms, browsers, and connectivity.

Large-scale on-demand mobile test services – This refers to the service techniques responding to on-demand mobile testing requests in mobile testing environments, test-ware, and test execution and control.

Multi-tenancy and customization support – This refers to testing multi-tenanted functions, behaviors, and QoS requirements for SaaS multi-tenancy. In addition, mobile TaaS supports customizable large-scale data load, traffic load, and user accesses with test service.

Contracting, utilization, billing and reporting – This refers to the service business model, pre-defined utility model, and cost metrics. Mobile testing clients will be charged using the *pay-as-you-test* utility model for their used mobile computing and testing resources on cloud.

Control and configuration of mobile test environments (general) – This refers to provide the required common-ware to support automatic mobile test management, test control, and testing interactions underlying mobile cloud. Diverse computing resources and test-wares can be selected, configured, and provisioned (or de-provisioned) dynamically.

Easy to test large-scale interoperation ability, mobility and connectivity (general) – This refers to some distinguish mobile features on large-scale cloud. Mobility testing verifies the quality of location-based system functions, data, and behaviors. Connectivity involves diverse mobile wireless networks which support the connectivity needs of mobile applications. Connectivity also affects the application performance and interoperability.

Test tracking, monitoring and coverage analysis (general) – This refers to tracking, monitoring, and coverage analysis techniques and solutions for mobile test operations at different levels for mobile apps and mobile web applications.

2.3. The test process of cloud-based mobile TaaS

Fig. 1 shows a test process (function testing) for mobile TaaS and conventional mobile testing. Compared to conventional mobile testing, the test process of cloud-based mobile TaaS primarily focuses on typical features such as tenant-based testing and scalability testing. Please note the testing activities in the dashed frame

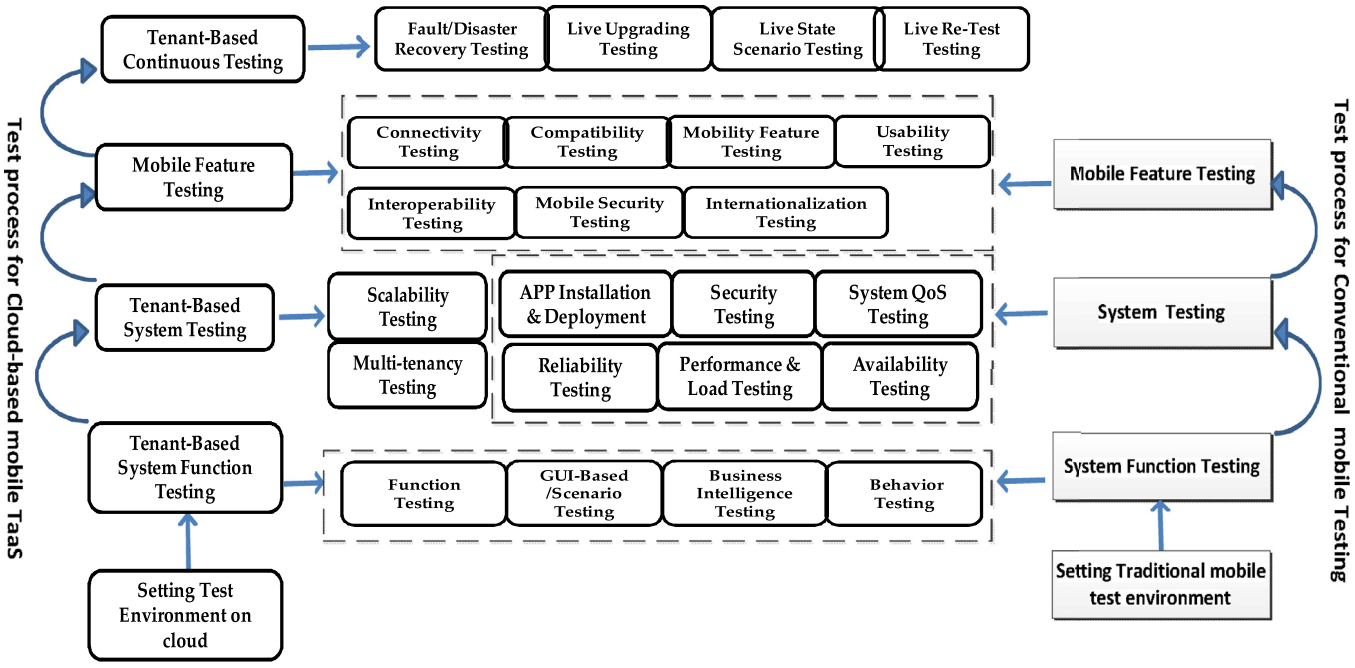

Fig. 1. A comparison of test process for cloud-based mobile TaaS and traditional mobile testing.

are shared by both sides. The testing process for cloud-based mobile TaaS shown in Fig. 1 includes the following five steps.

Step 1- Setting up mobile test infrastructure and environment and on cloud, including test control and run, tracking and monitor, interactions with TaaS server as well as its underlying mobile emulation cloud (or device cloud). This paper focuses on infrastructure service.

Step 2- Tenant-based system function testing includes diverse tenant-based service functions and features, such as function testing, GUI-based testing, behavior testing, and etc.

Step 3- Tenant-based system testing checks multi-tenancy, QoS, scalability, and etc.

Step 4- Mobile feature testing targets at mobile feature testing, usability testing, compatibility testing, and etc.

Step 5- Tenant-based continuous testing focuses on how to keep continuous validation for mobile system, including recovery testing, live upgrading testing, live regression testing, and etc.

In addition, compared to conventional mobile testing, cloud-based mobile TaaS has a number of new characteristics as shown in Table 1.

3. Infrastructures for cloud-based mobile TaaS

Since the conventional testing approaches cannot deal with the new requirements and features, mobile TaaS encountered many difficulties, such as large-scale testing services and on-demand testing needs. Thus, we need new approaches and infrastructures based on cloud to address the special features. This section presents and compares the current different infrastructure approaches. As shown in Fig. 2, there are three different mobile TaaS infrastructures on cloud instances. Here we discuss these infrastructures respectively.o ***Emulation-based or simulation-based mobile testing infrastructure on clouds***– In this form, mobile-based applications or SaaS instances on a cloud are validated using large-scale mobile emulators or simulators on cloud as shown in Fig. 2(a). This mobile cloud needs to support configuring diverse mobile emulators with different configurations. In addition, this cloud is required to have several key components, such as mobile emulation controller and test connection manager.o ***Crowd-based mobile testing infrastructure on clouds*** **–** In this form, mobile application servers on a cloud are validated using Ad-hoc mobile testing environment and TaaS infrastructure based on crowd sourcing as shown in Fig. 2(b). This approach does not need costs on mobile devices. In addition, it is easy to support large-scale compatibility testing and usability tests crossing diverse mobile devices. Some practitioners already applied the crowd-based testing approaches into businesses.o ***Device cloud-based mobile testing infrastructure on clouds*** - In this form, real mobile devices are purchased, deployed, and used to validate mobile-based software applications (including mobile APPs and mobile Web applications) and mobile SaaS as shown in Fig. 2(c). Unlike other mobile devices, they are structured, connected, configured, and set up to meet mobile testing service needs according to on-demand test service requirements.

The comparison among these three mobile TaaS approaches is presented in Table 2.

Currently, there are more and more industry practices on how to provide testing services for mobile applications based on (mobile) cloud. For instance, TestDroid by Bitbar offers on-demand mobile testing services on thousands of real Android and iOS devices (http://testdroid.com/) based on self-developed cloud infrastructure. It can get agile mobile development and testing process with continuous integration. Another well-known company uTest is based on the idea that crowdsourcing is a critical complement for testing web and mobile app in the lab (http://www.utest.com/company). It was designed as a testing community including courses, forums, tool reviews, etc., in order to provide customers with an end-to-end service offering, such as functional, usability and load testing for both IOS and android-based mobile system. *Testin* is currently another growing cloud-based app auto-testing service provider (http://mtestin.com/). Recently in 2015, the company released the crowd sourcing testing service platform in order to meet the demand of large-scale testing in cloud. *Yiceyun* developed *by neusoft* provides device cloud test automation services for android-based mobile systems. It supports the opensource framework of Robotium, Athrun, and Guerrilla (http://www.yiceyun.com). *Perfectomobile* provides MobileCloud

Table 1
A comparison between conventional mobile testing and cloud-based mobile TaaS.

	Conventional mobile testing	Cloud-based mobile testing as a service
Primary objectives	Validate the quality of mobile APPs on mobile operation environments of specified mobile devices or from different web browsers.	Provide on-demand testing services on cloud; Leverage a scalable mobile testing environment To assure pre-defined given QoS and SLAs.
Testing focuses	Diverse software errors in its structures, functions, behaviors, user interfaces, and connections to the external systems; System non-functional requirements such as performances, reliability, availability, vertical scalability, security, and etc.	Multi-tenancy, customization, and configurability; Mobile SaaS scalability; Connectivity to its external contexts; Interoperability and portability
Testing execution	Offline testing in a test laboratory before a product delivery.	Offline testing in a private cloud-based test environment; On-demand test execution in a cloud-based virtual test environment; Continuous testing for SaaS in/on/over clouds.
Testing environment	A pre-configured test environment in a test laboratory with purchased hardware/software and tools.	A scalable mobile test environment based on cloud with diverse computing resources and tools; Supporting web browsers on different mobile platforms and devices.
Testing process	Enterprise-oriented test processes for each project.	Crowdsourcing based process; Well-defined TaaS processes by TaaS vendors; Emphasis on tenant-based testing, service component testing, etc.
Testing techniques	Apply selected well-known white-box and black-box testing techniques at the component level (or unit level) and the system level.	Required innovative continuous testing techniques; New testing solutions to deal with multi-tenancy and elasticity.
Testing tools	Use limited testing solutions and tools with the purchased licenses.	On-demand usage of diverse test tools with shared licenses in a cloud environment based on pay-as-you-use.
Project cost	Required hardware/software (license) costs in a test lab, plus engineering costs.	Based on a pre-defined SLA; Pay-as-you-test service costs.

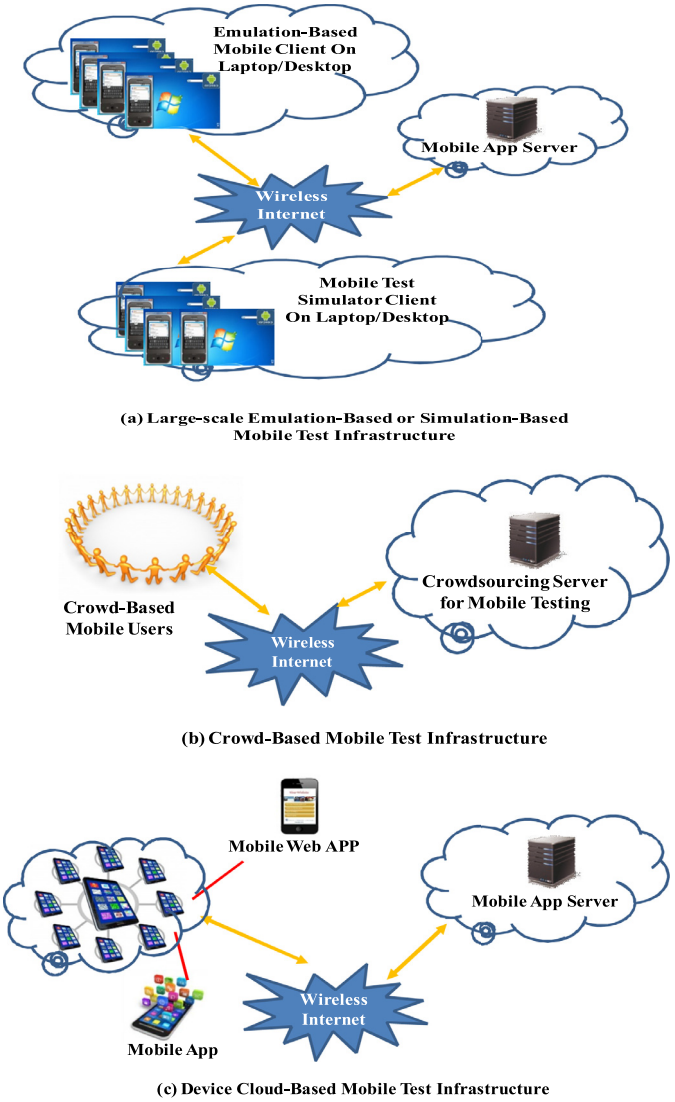

Fig. 2. Samples of mobile TaaS infrastructure on cloud instance.

Interactive, MobileCloud Automation, and MobileCloud Monitoring (http://www.perfectomobile.com/).

Here two industrial cases of best practices in mobile TaaS are shown. Humana, a fortune-100 health and well-being company with more than 40,000 associates provides consumers and businesses with access to information via mobile channels across their various health care operations in all 50 U.S. states. Humana required a solution for testing mobile apps and websites, to ensure their members continued reliable access to healthcare information. With the support of DeviceAnywhere mobile testing platform from Keynote, Humana has set up a local dedicated testing environment to cover all of mobile channels. With this test environment the healthcare provider can run manual and automated tests on 50 live devices, including iOS, Android, and Blackberry platforms. In the last thanksgiving, some businesses performed very well with their mobile apps. Internet Retailer cited Catchpoint data that pointed to the five native mobile apps that provided users with excellent buying experiences on Black Friday. At No. 1 comes Foot Locker with search times as fast as 0.133 seconds with performance testing from Perfecto (http://blog.perfectomobile.com).

4. A developed cloud based mobile TaaS system (MTaaS)

We are developing a mobile TaaS system, including both IaaS and PaaS pieces. Currently, we have developed a cloud-based mobile TaaS system (MTaaS) as *infrastructure-as-a-service* (IaaS) for mobile app testing known as IaaS piece. The PaaS piece (such as test automation platform) is still under work. MTaaS supports for several key infrastructure services such as resource provisioning, monitoring and billing services. To provide mobile IaaS, MTaaS manages massive resources and has responsibility of provision, allocation, and charging. Server, emulator, device, and mobile hub are the key resources in MTaaS. This section describes the infrastructure of MTaaS, the detailed system design, the provided service functions, and resource allocation algorithms. In addition, system evaluation is discussed in the end of this section.

4.1. Infrastructure of MTaaS system

The developed system MTaaS provides IaaS for mobile testing with setting up a mobile infrastructure, which supports provisioning, management, monitoring and billing services. The

Table 2
Comparisons of cloud-based mobile TaaS approaches.

Perspectives/ Different approaches	Emulation-based testing on clouds	Mobile testing in crowdsourcing	Device cloud-based mobile testing
Mobile TaaS service model	Emulation-based mobile TaaS service model	Crowdsourcing-based service model	Remote mobile TaaS service model
Business models and billing	Pay-as-you-use for device emulators and other testing services	Crowdsourcing cost models	Pay-as-you-use for remote devices and other testing services
Mobile testing environment	Emulation-based mobile taas infrastructure	Ad-hoc mobile testing environment, and TaaS Infrastructure	Shared Mobile TaaS infrastructure
Mobile devices costs	Only use emulators, no device costs	No costs on mobile devices since it uses mobile devices from crowded testers	Device rental costs
Mobile end-to-end transaction testing	Emulation-based end-to-end transaction testing	end-to-end transaction testing in crowdsourcing	Large-scale device oriented end-to-end transaction testing
Mobile usability testing	No coverage on real mobile user experience	Easy to support usability testing	Scalable test coverage on real mobile devices
Mobility testing	No reallocation service testing, Using emulators only	Easy to performlocation service testing by crowded testers	Limited location service testing
QoS Testing for scalability, performance, reliability and availability	Emulation-based QoS testing at the limited scale	Ad-hoc QoS testing using crowdsourcing; low testing quality risk; an uncertain validation schedule	Large-scale QoS testing
Mobile Security testing	Emulation based security testing only	Ad-hoc security testing with risk problem	Diverse device based security testing
Mobile app function and gui testing	Emulation-based Testing, hard to test functions related to real device	Ad-hoc mobile testing based on No. of users	Diverse device based function testing

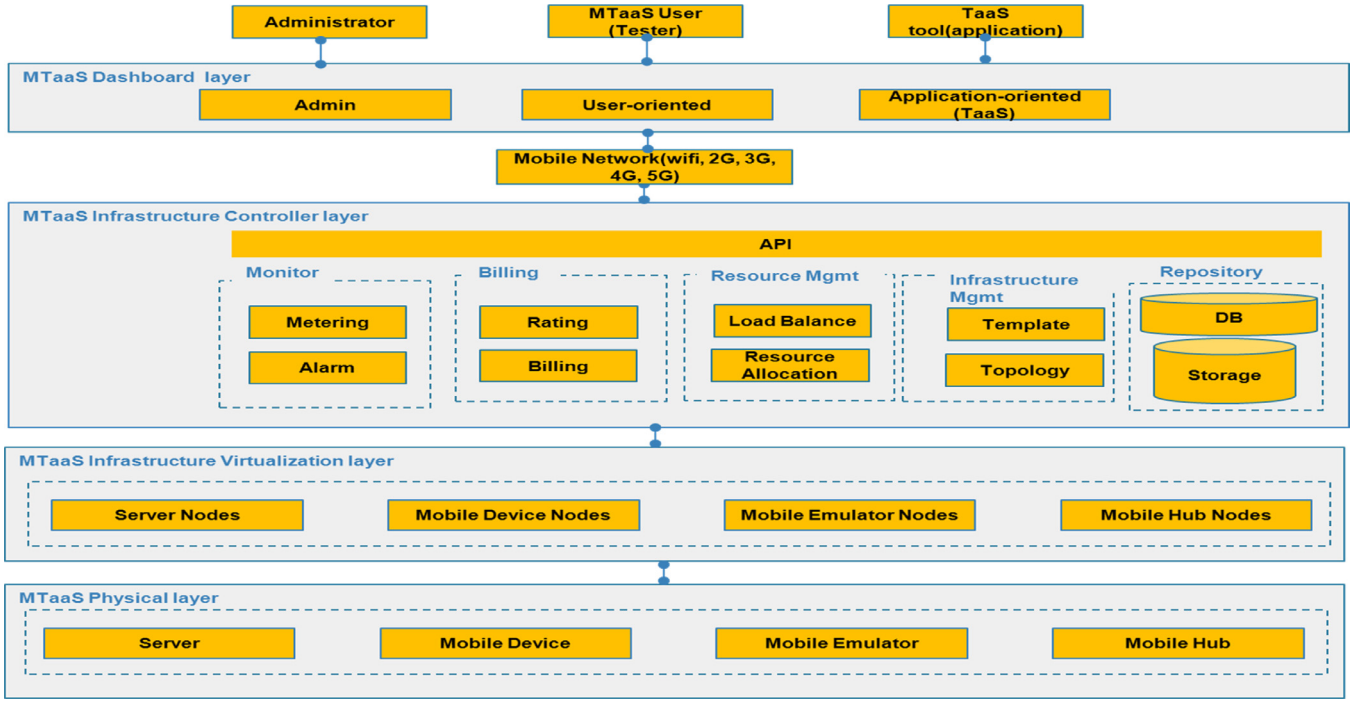

Fig. 3. The architecture of MTaaS.

approach to the infrastructure is based on *device cloud* as introduced in Section 3. Emulation cloud is also supported in MTaaS. The system aims to provide an on-demand mobile infrastructure for mobile app testing clients' access to any mobile infrastructure such as mobile devices, emulators, mobile hubs and server machines, in order to deploy, host and test mobile applications. The primary system clients include Admin, users (such as testers), TaaS tools or applications (They configure the provided IaaS resources for future testing). Fig. 3 shows the infrastructure of MTaaS. There are four layers in the architecture, including *dashboard, infrastructure controller, virtualization, and physical layer.*

The *dashboard layer* provides a detailed view of the available resources to MTaaS clients. Also, a graphical representation is presented to facilitate users to monitor their resources running in MTaaS.

The *controller layer* consists of several key components, such as monitor, billing, infrastructure management, etc. For instance, billing is the fundamental feature for service providers to operate this infrastructure. Consumers can set up their payment account, query the bills, and make payments. Various cloud systems or tools can be used. In MTaaS, we adopt OpenStack as cloud-based infrastructure management support. OpenStack is a well-known open-source cloud operating system for managing large pools of compute, storage and networking resources throughout a data center, managed through a UI dashboard or via the OpenStack API (Openstack documentation 2016). There are several interactions among the internal APIs and external APIs. We have developed our own APIs, which use RESTful web services to interact with the system. For multiple requests, the resource allocation and management engine employs a load balancer algorithm to distribute the workload among multiple hosts spread across different clouds. Our

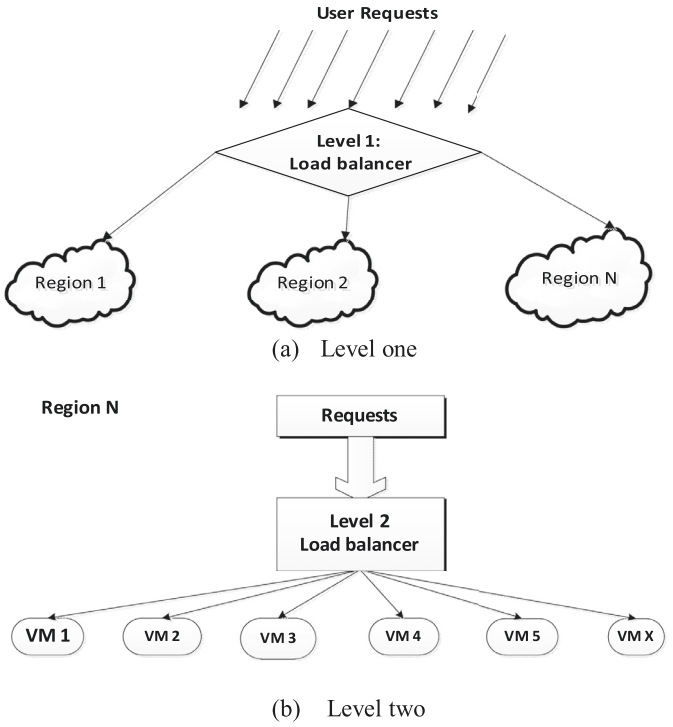

Fig. 4. The two-level virtualization design in MTaaS.

approach aims to achieve load balancing of clouds with *minimum resource starvation* and *maximum resource utilization* by the means of *optimized resource provisioning*.

The *virtualization layer* separates physical infrastructures to create various dedicated resources. It is the fundamental technology that powers cloud computing (Mann, 2015). Relying on virtualization technology, a host on a physical machine can start multiple virtual resources. A large number of virtual resources are the basis for IaaS. Due to the various performances, the used virtual solutions are different, resulting in cloud computing resources in heterogeneous characteristics.

We proposed a two-level virtualization solution for mobile cloud. Level one is in front of OpenStack regions, and level two is in front of Virtual Machines. Fig. 4 shows the sample of two-level virtualization design. As shown in the Figure, Load balancer is a crucial component inside of IaaS since massive requests from customers need to be handled, distributed, and processed. Imaging multiple regions available, load balancer at this level distributes users' requests to different regions, as shown in Fig. 4(a). Load balancer running at level one works closely with OpenStack Controller node that commands the whole functionality of scheduling jobs in each region. The interface between load balancer and controller is based on RESTful APIs which provide plentiful runtime information of system. Behind level one load balancer, another important node is virtual machines distributor. It functions load balancing as level two shown in Fig. 4(b). Virtual machine is the fundamental resources in each region, and the host of all the virtual mobile devices. When user requests are forwarded to one particular region where the virtual machine load balancer is running, they are distributed further to different virtual machines.

The *physical layer* represents the physical resources such as mobile devices, servers, emulators, and hubs of our own or donated by providers.

Fig. 5 presents the cloud-based infrastructure in MTaaS. The remote mobile emulation cloud and mobile device cloud are both adopted to support large-scale mobile testing. The IaaS server handles diverse on-demand testing services. The IaaS management server controls the infrastructure service process and business flow.

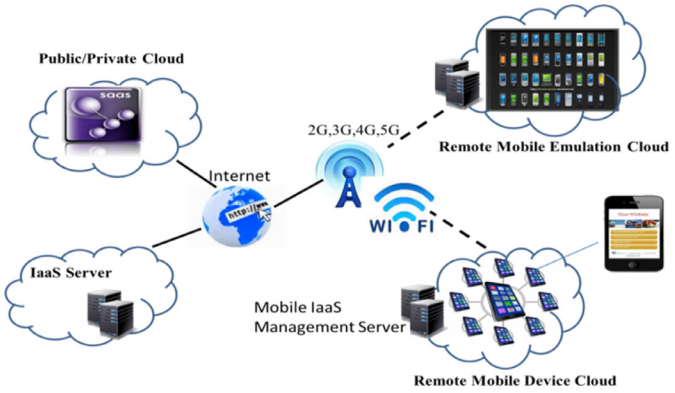

Fig. 5. Cloud-based infrastructure of MTaaS.

4.2. MTaaS system design

This subsection describes the system design primarily in user Interface, database, algorithms, and MTaaS connectivity.

User interface design: Various types of clients like *admin*, users, or applications (TaaS tools) can access the system. Depending on the user type, *Visualization* is presented to the user. *Admin* can have access to run time graph visualization. The request generation UI presents the user to specify resources to the requests. This further is handled by proxy server. The billing UI is made available to user as soon as resources are granted to/ registered by users. In short, users interact with the UI through web browser. A user can perform various relevant actions on UI such as requesting resources through request loader module. The incoming requests are accepted by a proxy server ('Load balancing server'). This server is Node.js server. The navigation flow diagram shown in Fig. 6 represents the flow between web pages and explains the to and fro movement in UI application. The main page is home page. Users can navigate from home page to 'request generator' or 'billing' or 'resource usage' pages. The additional *run time graph* page is also visible, pictorially representing resource status from admin's point of view.

Logic and algorithms design: In MTaaS, resource management is an important part in controller, mainly including load balancing and resource allocation. The incoming requests are firstly distributed to a load balancer port based on diverse load balancing strategies. They are directed to servers depending on the chosen strategy. Then, the physical and virtualized resources need to be allocated. In mobile device cloud, there are multiple cloud data centers across various locations. Based on the requests' static or dynamic nature, various types of resource allocation algorithms to support the nature of requests are needed. Thereby request handling, resource allocation, and load balancing go hand in hand in the system. The detailed strategies and adopted algorithms for resource allocation are discussed in Section 4.4.

Database design: The system is facilitated by HTML 5 – local storage feature as it allows fast access to data. At initialization, the data is fetched from My SQL and stored in HTML5. Then all the storage and retrieval is performed through HTML 5. The database design has been made taking into account normalization rules to minimize redundancy. The design facilitates minimal storage space constraint and is optimized to enable faster date access and recovery. A sample piece of data in the application is shown below.

Server {

'Server Name': 'Test Server 3',

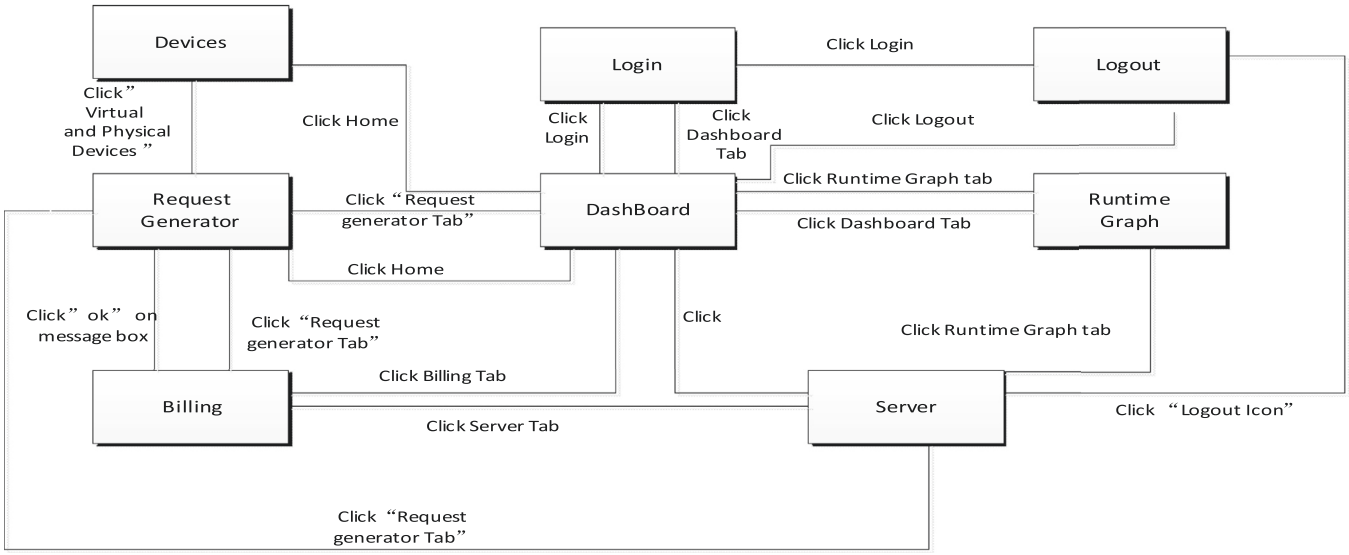

Fig. 6. The navigation flow of UI.

' IP Address': ' 10.0.0.2',
'Image Name': 'cirros-0.3.2 × 86_64-uec',
'Location': 'China',
'Status': 'Active',
'Date Created': '6/20/2016',
' Availability': ' Unavailable'}

It is related to server information at a given point of time in system. The data is stored in local storage and can be fetched from it.

MTaaS connectivity design: A set of RESTful APIs have been used to establish communications between user interface and application logic layer. The clients from any browser can interact with the system. The interactions are carried out through RESTful API calls (GET, PUT, POST, and DELETE). When the method type is GET, data associated and pertaining to desired target as designed is fetched. We have developed our own APIs, which use RESTful web services to interact with the system. Partial designed APIs are presented in Table 3 with a sample request for each. For instance, *Create Cloud* is a POST method to create a cloud in a particular location. *Create Resource Allocation Request* is a POST method to create a request for resource allocation to one of the resource types, i.e., Mobile Device, Mobile Hub, Emulator, and Server Machine. *Get Cloud By Name* is a GET method, which retrieves a cloud and hosts that are present along with usage statistics of the cloud, and *Delete Server* is a POST method, which deletes a particular instance of host.

Table 3
Samples of designed RESTful APIs.

RESTful API calls	API functions
POST	*Create cloud*
POST	*Create host*
POST	*Create resource allocation request*
GET	*Get cloud by name*
DELETE	*Delete server*
POST	*Setup cloud*
POST	*Request loader*

4.3. Service functions of MTaaS

The developed MTaaS system provides several service functions, including request generator, resource provisioning, dashboard monitoring, mobile network connectivity, and billing service. Next, those service functions are explained respectively.

Request loader service: The request loader generates multiple mobile-enabled service requests. The incoming mobile requests demand diverse mobile cloud resources such as mobile devices, mobile hub, emulators and server machines to form a virtualized mobile testing infrastructure. MTaaS provides a convenient way to manage these emulator resources with operations of creating, querying, and deleting. Users can create a new emulator with a few clicks. Before submitting, users can modify the parameters of emulators to satisfy their test requirements. Request loader service function facilities our testing needs through a number of generated requests.

Resource provisioning and management service: This component allows an administrator to apply different load balancing optimization algorithms as part of optimized resource allocation strategy to deliver selected mobile testing infrastructure requested by users to form a mobile cloud testing environment. In addition to Mobile Devices, Mobile Hubs, Emulators and Server Machines, components such as Mobile Hub Manager, Emulator Manager, and Device Manager are also provided. They control and manage a number of configurable mobile devices and emulators. Users can query, choose, and terminate resources.

MTaaS provides a nice virtual device screen function by which users can manipulate their remote emulators as same as local ones. Besides virtual devices screen feature, users also can log into their remote emulators via standard SSH tools. Devices are another kind of essential resources provided by MTaaS, since most mobile developers need to verify some features with real smart devices. MTaaS provides the same interface by which users can connect the popular test tools and mobile app IDE to their devices in the cloud. Users can quickly create any model of devices that they need in a minute with MTaaS, and don't need to spend a huge money to buy them. Fig. 7 shows the screenshot of configured and selected emulators and physical devices. The left screenshot in Fig. 7 shows the requested emulators, including their name, OS type and version, RAM, and status. The right screenshot presents the information of requested physical mobile devices. In addition, in MTaaS, a user can wishfully lease resources, and in turn gain monetary advantage for that. Users can provide resource donations such as servers, devices, and hubs.

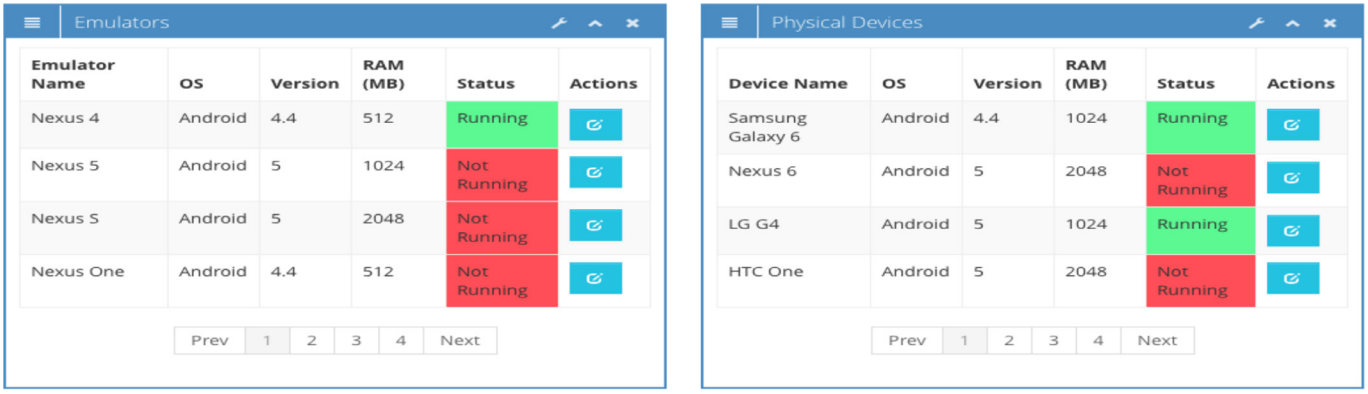

Fig. 7. Screenshot of emulator and physical devices.

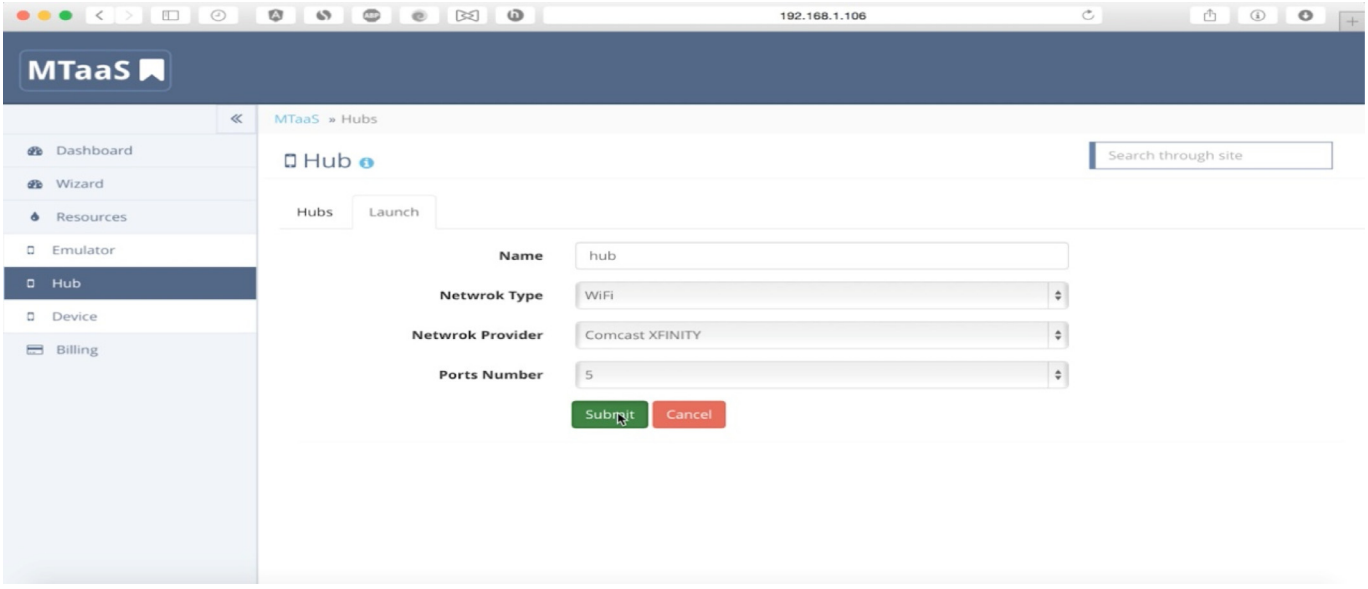

Fig. 8. Connecting MTaaS to network by using hub.

Mobile network connectivity service: Mobile hub provides network connectivity for both emulators and devices. Most of the mobile apps are internet-based and very sensitive to network bandwidth, latency, and throughput. Mobile hub can satisfy these kinds of mobile app with multiple choices of network connection like *wifi, cellular*, etc. Each bub is combined with several mobile devices or emulators. As shown in Fig. 8 users can create, query, and delete mobile hub according to their requirements. Once a mobile hub is created, users can attach both emulators and devices to this hub so that all those equipment get network connection.

Through resource provisioning and network connectivity, the result of requested infrastructure is configured. Fig. 9 shows a sample runtime graph of the infrastructure configuration in dashboard**.** A hub is connected to Test Server 1 with several mobile devices of Android OS in Location of USA. Fig. 9 also shows the configured clients' infrastructure needs through real physical resources plus software controlling and defining.

Billing service: With this module, a business cost model is implemented with billing metrics to determine the costs for users based on their service request types. In MTaaS, we propose a systematic cost model for mobile testing IaaS, PaaS, and SaaS. For IaaS, the primary cost factor includes the number of CPU cores, the amount of RAM and disk storage, as well as the mobile resource usage. For PaaS level, the cost factors are the test environment, usage time, and configurable testing platform. For SaaS level, SaaS testing tools, cost sharing, and license are considered as the main cost factors. Billing service function provides a fundamental support for service providers to operate MTaaS. Both users and administrator can leverage this feature. Administrator manages all the user payment accounts and bills. Users can choose their billing plan within "pay as hour go" or "month flat rate" . As shown in Fig. 10, a billing query function facilitates users to view all the bills in history.

Mobile monitoring service: The UI dashboard provides monitoring and reporting functions including resource usage across different clouds, usage index per cloud, the utilization and availability of CPU cores, RAM and disk storage. For each virtual resource when created, the system will instantiate a VM monitor object for real-time monitoring of the indicators of the VM parameters at runtime for a single request in order to get access to the VM load balance information. Besides monitoring resources, dashboard also provides a quick start for users who firstly log into MTaaS. Users can pick-and-choose different plans based on their requirements. Fig. 11 shows the screenshot of the monitoring services in MTaaS dashboard. There are some emulator plans and device plans requested from users. Here *Emulator plan A, Emulator plan B, Device plan A, and Device plan B* are available. Through the use of highly interactive D3 JS library in java script, we provided graph visualization to all clients to know the current status of overall resources used.

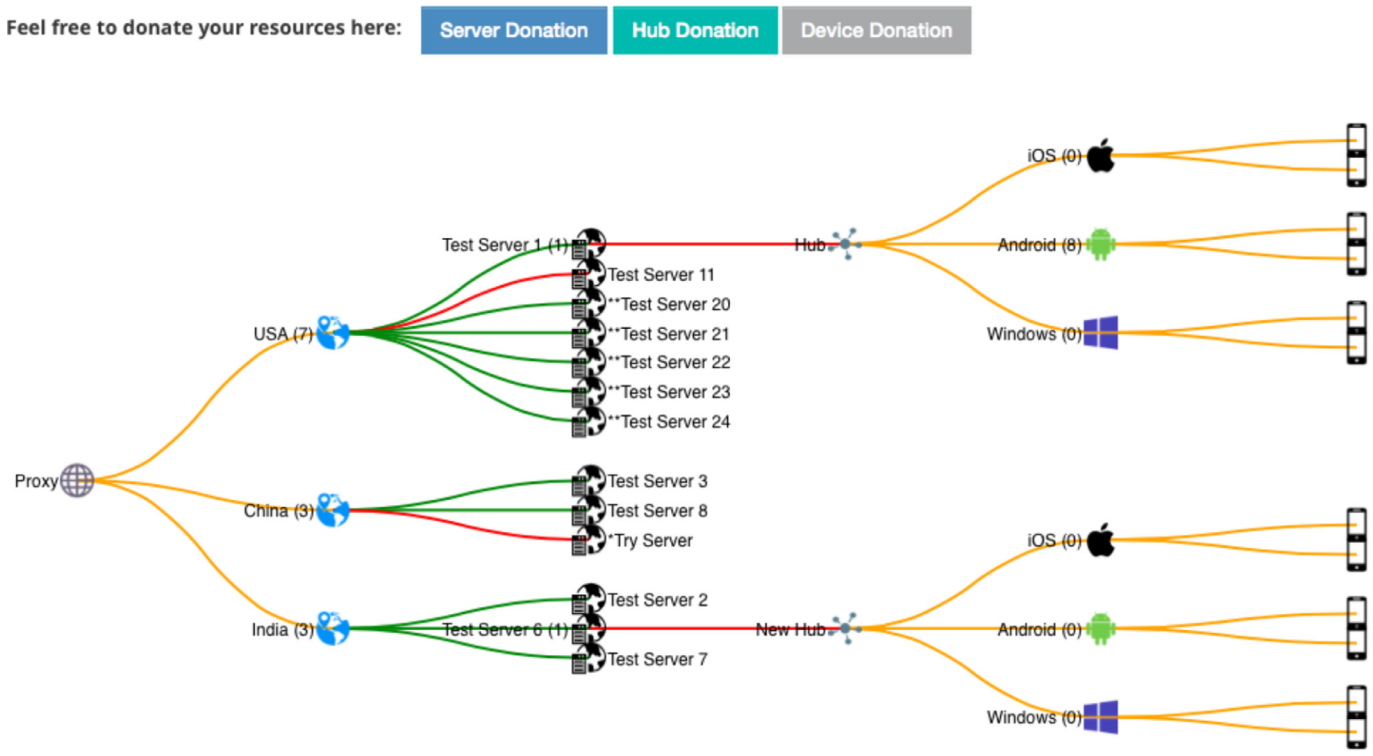

Fig. 9. The dashboard runtime graph of infrastructure configuration.

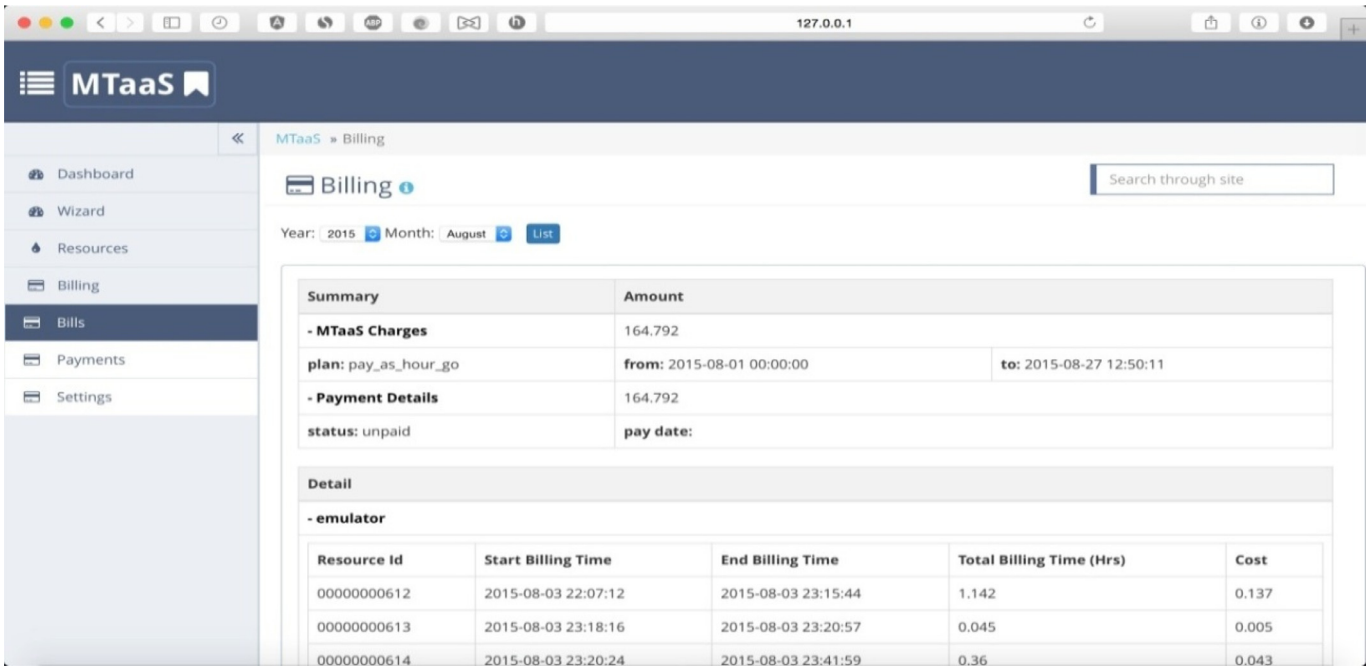

Fig. 10. A service billing screenshot.

4.4. Resource allocation in MTaaS

Resource allocation has a significant impact in cloud computing, especially in pay-per-use deployments where the number of resources is charged to application providers. The issue here is to allocate proper resources to perform the computation with minimal time and infrastructure cost. Proper resources are to be selected for specific applications in IaaS (Zhan et al., 2015; Manvi and Shyam, 2014; Yehuda et al., 2014; Do and Rotter, 2012). In MTaaS, how to utilize the traditional resource allocation algorithms to address the new needs in mobile device cloud effectively raises new issues and challenges. In the following subsections, the issues, resolutions, and proposed algorithm are discussed in detail.

4.4.1. Issues and resolutions

Issues:

A lot of differences are exited between conventional cloud and mobile device cloud. The conventional cloud focuses on computing

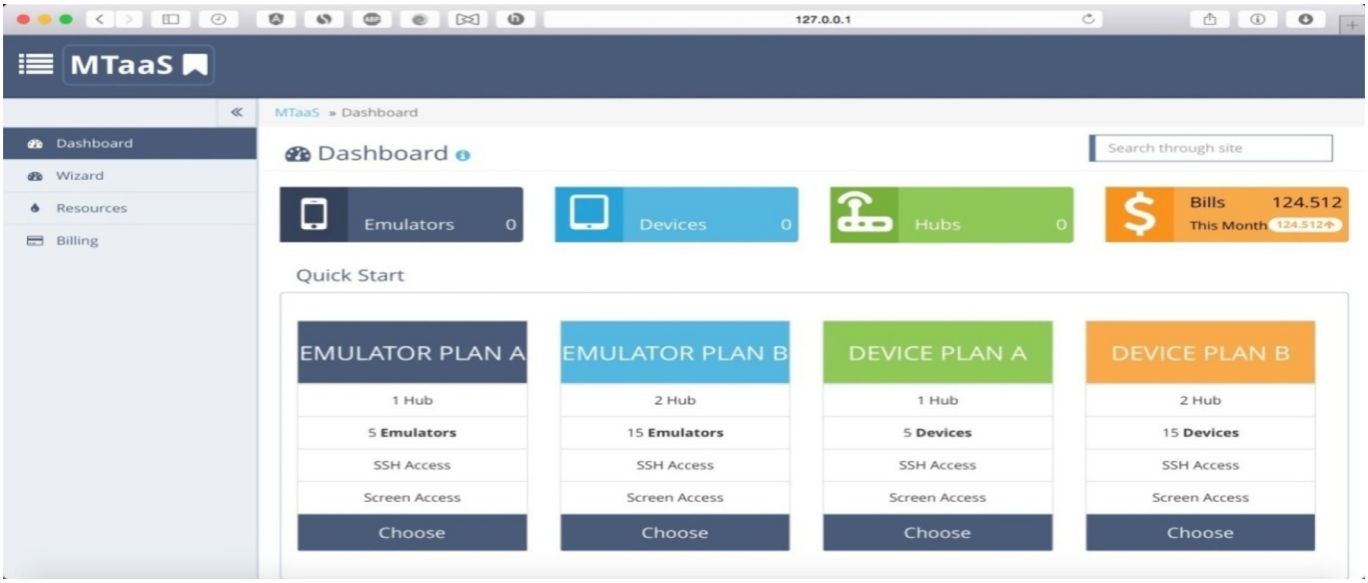

Fig. 11. Dashboard UI of monitoring service.

and storage while mobile resources and connectivity are paid more attention in mobile device cloud. For instance, the primary new resources in mobile cloud are mobile devices and hubs. In addition, mobile network connectivity is performed through various wireless solutions, such as 2 G, 3 G, wifi, cellular, etc. All these features bring new issues for cloud-based mobile TaaS.

In MTaaS, cloud computing infrastructure services are a collection of resources, cloud resources diversity and heterogeneity. Current device cloud can be provided by large business companies. Thus, we assume that the devices are from three parts: users' devices, test lab provided devices, and devices in MTaaS. First of all, basic service resources can be divided according to geographical area. A region is shown in the resource layer structure. Each region contains a DC (data center), and each DC is considered as a cluster system, with multiple physical machines. The diversity of devices and users lead to location issue for resource allocation.

According to our survey, there are a number of published algorithms applied in resource allocation in cloud computing (Mann, 2015; Zhan et al., 2015; Manvi and Shyam, 2014; Yehuda et al., 2014; Do and Rotter, 2012), such as FCFS, Round Robin, GA, ACO, PSO, etc. However, diverse mobile resources, different locations, and large-scale requests lead to the need of revised or new algorithms in MTaaS. The nature of incoming requests in MTaaS can be static or dynamic. The pre-booking or planed requests are static while the on-demand or real-time requests are dynamic. The purpose of applied algorithms for static requests aims at optimization, and for dynamic requests the on-demand features need to be addressed. Thus, diverse algorithms need to be selected to deal with the static and dynamic natures of different requests.

Resolutions:

To meet the demand of various requests, we need to take advantage of the available algorithms. In MTaaS, we proposed a *Hybrid Algorithm* for resource allocation. *Round Robin algorithm* and *Random Algorithm* are selected for dynamic requests while *Ant Colony Optimization* is selected for static optimization. For large-scale requests, we adopt *request peeking* to deal with request priority issue. In addition, a Location Aware Algorithm is proposed to deal with the location issue. The overall goal of *Hybrid Algorithm* aims to improve the system performance while reducing costs due to wireless connectivity, communication, and diverse locations. Next, the proposed *Hybrid Algorithm* is discussed in detail.

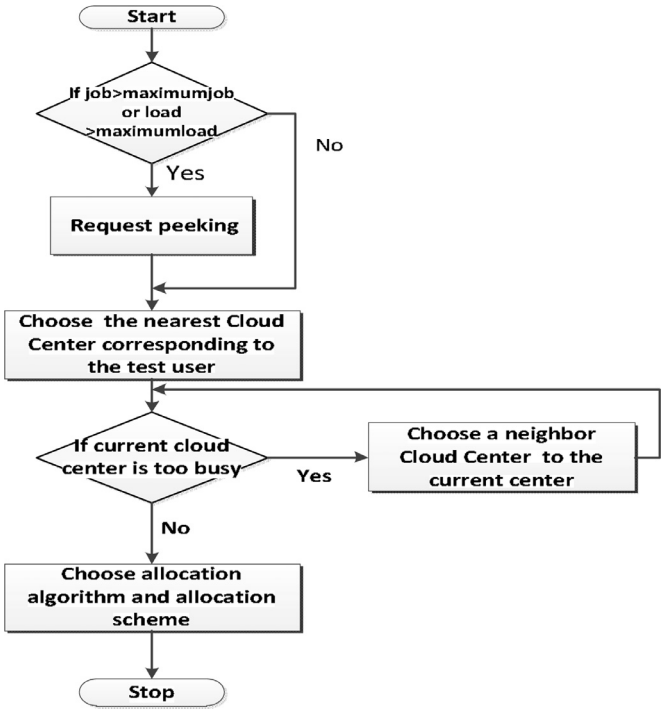

Fig. 12. Overview of hybrid algorithm.

4.4.2. The proposed hybrid algorithm

Fig. 12 shows the overview of the algorithm. The key step is request peeking, location aware, and algorithm selection. The three steps are described below in detail.

Step 1: Request peeking. An approach to request optimization is implemented through request peeking. At any point of time, the set of request(s) reside in a request queue after submission. On turning on this feature, the requests are served on basis of priority. The priority can be based on request duration time, urgency degree, service cost, request level, and etc. Fig. 13 presents the process of request peeking. Once the requests are accepted, the requests in queue are sorted according to the priority rules. The number of peeking can be set in terms of practical needs.

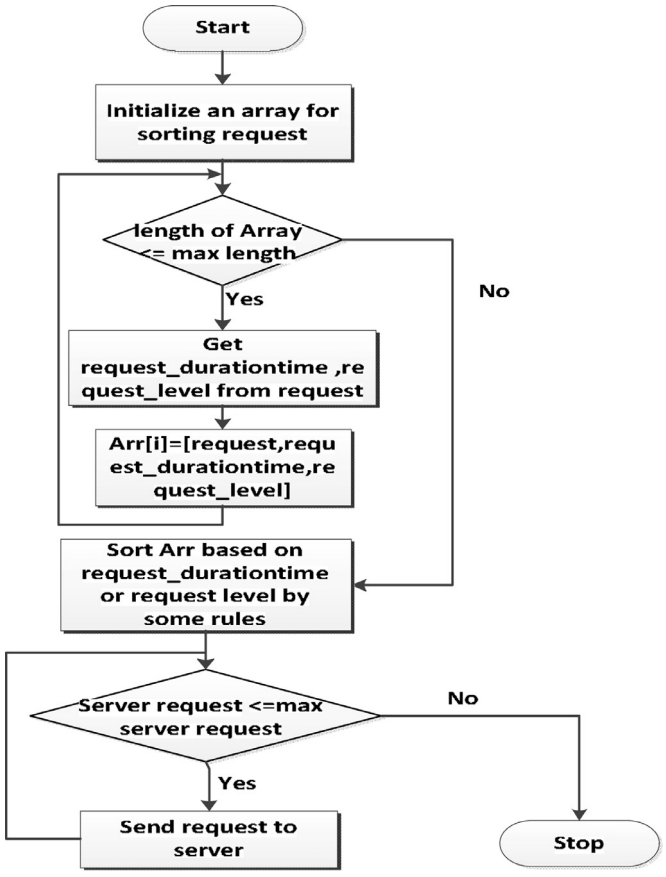

Fig. 13. The process of *request peeking*.

Step 2: Location aware. In reality, a cloud computing system usually consists of some data centers which are distributed in different geographical areas. These data centers connect with each other by dedicated network with high reliability and high transmission rate. Compared to traditionally centralized internet data center (IDC), users in different regions can access to the close data centers in MDC environments. The services for users requested can be provided by data centers nearby, as it can reduce access latency and network load. Meanwhile, location aware improves communication efficiency and supports to reduce network costs for some high-bandwidth applications. Therefore, for serving diverse nature of requests, we proposed *Location aware algorithm for MTaaS*. The pseudo codes for *Location-aware algorithm* are presented in Algorithm 1. In the algorithm, resources are distributed based on the location distance between user and data center. In our implementation, test center and user regions are the key location factors. Firstly, the users' location distance is calculated, and then the nearest cloud data center is computed. Finally, the users' requests are allocated to the center.

Step 3: Algorithm selection. In order to continuously optimize the performance of the implemented system, we analyze several algorithms for resource allocation. As we discussed, once the requests are submitted by user, they are received by proxy server. One of the algorithms picked for requests come into consideration then. We have selected three typical resource allocation algorithms. These are used for serving dynamic as well as static nature of requests. The first algorithm is *Round Robin* (RR). In RR algorithm, time slices are assigned to each process in equal portions and in circular order, handling all processes without priority (also known as cyclic executive) (Hirenkumar, 2015). RR scheduling is simple, easy to implement, and starvation-free. The second

Algorithm 1. Location aware algorithm for MTaaS.

Input: undetermined request queue
Output: request scheduling policy
1: FOR request IN Arr do
//compute nearest cloud center for each request in queue
2: Get coordinates of cloud centers list;
3: Users_location = compute_location(request);
//current request user's location
4: WHLIE list != NULL
5: Current_center = list.pop;
6: Cloud_location = compute_location(Current_center);
7: Distance=compute_distance(users_location,cloud_location);
//compute the nearest cloud center
8: IF distance < min_distance THEN
9: nearest_cloud = Current_center;
10: min_distance = distance;
11: END IF
12: END WHILE
13: Allocate request to Current_center;
//assign each request to the nearest center
14: END FOR

one is *Random Algorithm*, which is a commonly-used resource allocation strategy. In MTaaS, mobile hubs and devices are combined together. The complexity of nodes is not high. Thus, RR or *Random Algorithm* is suitable for resource allocation. The third algorithm is *Ant Colony Optimization* (ACO), which is inspired from the ant colonies that work together in foraging behavior (Dorgio and Birattari, 2010).

Our algorithm selection process is as follows. When the user quest is dynamic, we select simple RR or *Random Algorithm* to handle on-demand needs. When the user quest is static and predefined, ACO is selected for optimization of resource allocation. The detailed comparison of diverse algorithms for resource allocations are not discussed in this paper. In the future work, we plan to perform several comparison experimental studies to investigate the application effect.

In summary, the hybrid algorithm based on the three steps above is described in Algorithm 2 below. At first, Algorithm 2 deals with request peeking (lines 1–14) according to the setting rules; then, *Location-Aware Algorithm* is performed (lines 15–20); later, different algorithms are selected in terms of static and dynamic request nature (lines 21–27).

4.5. System evaluation

4.5.1. System testing

We performed both function and performance testing of MTaaS system. For scenario-based function testing, we designed a total of 29 test scenarios for dashboard, emulator management, device management, hub management, and billing management. All the test cases have been executed and passed. For example, Table 4 lists the five passed test cases for the emulator management function.

For performance testing, we mainly used the *average response time* and *error ratios* as the two indexes to demonstrate the performance with the tool *JMeter*. For instance, the typical five scenarios we chose are as follows. The testing results for the five scenarios are shown in Tables 5 –9 respectively. They are as follows.

Scenario 1: Consecutively increase user number from 0 to 1000, each user sends the request to get the responding emulators with running status.

Scenario 2: A user request to launch 20 emulators at one time.

Scenario 3: Consecutively increase user number from 0 to 20, each user sends the request to launch 5 hubs.

Algorithm 2. Hybrid algorithm for MTaaS.

```
Input: undetermined request queue
Output: request scheduling policy
1:  Arr=[]
2:  FOR request IN request list do
 //perform request peeking on request queue
3:  IF server.job > maximumjob or server.load > maximumload
4:  WHILE len(Arr) < max length do
5:  request_duration=request.getduration();
6:  request_level= request.getlevel();
7:  Arr.append (request_duration,request_level);
  //assignment for each request
8:  END WHILE
9:  Arr.Sort(); //sort by some policy
10: WHILE server.request < max request do
11:  Send request to server;
12:  ELSE pass;
13:  END IF
14: END FOR
15: FOR request IN Arr do
16:  perform Algorithm 1; // Location aware algorithm
17:  Requestlist=[];
18:  FOR get request from server do
19:  Requestlist.append (get request from server);
20: END FOR
21: FOR Requestlist != NULL do
22:  IF Requestlist[i]. expectedtime < d
23:  IF Requestlist[i]. nature is static
24:  Apply Requestlist[i] and Requestlist[i+1] to round robin
25:  and random;
26:  END IF
27:  ELSE Apply Requestlist[i] to ACO;
28: END IF
29:  Delete Requestlist[i];
30: END FOR
```

Table 4
Partial function testing results.

Test case	Description	Result
Func_emulator_01	Create a emulator	Pass
Func_emulator_02	Query emulators	Pass
Func_emulator_03	Delete a emulator	Pass
Func_emulator_04	View screen of a remote emulator	Pass
Func_emulator_05	Connect to a remote emulator via SSH	Pass

Table 5
Testing results of scenario 1.

Function	Get emulators of each user
APIs	/emulators
Method	GET
Users	Consecutively from 0 to 1000 in 5 min
Avg. requests	18 per sec.
Avg. response time	10 ms
Error ratio	1.08%

Scenario 4: Consecutively increase user number from 0 to 1000, each user sends the request to get the responding hubs with running status.

Scenario 5: Consecutively increase user number from 0 to 1000, each user sends the request to get the responding emulators and hubs with running status.

Fig. 14(a)–(e) presents the testing results of the five scenarios in a curve graph respectively. The left vertical axis represents the number of VU (virtual users). The right two vertical axis shows RT (response time) and the number of Hits. The horizontal axis represents the testing time. Fig. 14(a) shows the testing results of Scenario 1. As shown in the graph, the RT value keeps stable during the 5 min, and the average RT value is 10 ms. The number of *Hits/s Total* is fluctuant around 20 each second. Obvious *Hits Errors* appear six times in the duration. The value of *error ratio* is 1.08% as listed in Table 5. In Scenario 2, RT value is fluctuant from around 200 ms to 400 ms most of the time as shown in Fig. 14(b). Obvious *Hits Errors* happen three times in the duration. In Scenario 3, the number of *Hits/s Total* is one almost all the time in Scenario 3 shown in Fig. 14(c). Obvious *Hits Errors* only appears one time. The *error ratio* is 1.21% in a low level. The performance of Scenario 4 (shown in Fig. 14(d)) and Scenario 5 (shown in Fig. 14(e)) shows no significant difference as the test case is similar.

Table 6
Testing results of scenario 2.

Function	Launch emulators
APIs	/emulators
Method	POST
Users	1
No. of Emulators	20
Avg. response time	300 ms
Error ratio	2.17%

Table 7
Testing results of scenario 3.

Function	Launch hubs
APIs	/hubs
Method	POST
Users	Consecutively from 0 to 20 in 1.5 min
No. of hubs	100
Avg. response time	270 ms
Error ratio	1.21%

Table 8
Testing results of scenario 4.

Function	Get hubs
APIs	/hubs
Method	GET
Users	Consecutively from 0 to 1000 in 1.5 min
Avg. requests	20 per sec.
Avg. response time	125 ms
Error ratio	4.68%

Table 9
Testing results of scenario 5.

Function	Get emulators & hubs
APIs	/emulators, /hubs
Method	GET
Users	Consecutively from 0 to 1000 in 1.5 min
Avg. requests	20 per sec.
Avg. response time	200 ms
Error ratio	4.83%

In summary, the RT value is less than 300 ms for all of these tested scenarios. Thus, according to the testing results, the system is doing well in performance for RT. The *error ratio* is relatively higher when the number of virtual users is increasing to 1000 in a short time, such as 1.5 mins in Scenario 4 and 5. The testing results could support us to find more issues of the system performance bottleneck.

4.5.2. A comparative study between MTaaS and other systems

Currently, there are not any open-source implementations for mobile TaaS according to our survey. Various online mobile testing platforms are available nowadays. However, their functionality is limited since they only provide one or few of the testing mechanisms. The related business players such as *Perfecto Mobile* and *uTest* (introduced in Section 3) provide partial features such as cross platform testing, test simulation, and crowdsourcing testing.

Fig. 14. Performance testing results of MTaaS system.

Different from the current business tools, the developed system in this paper aims to provide *infrastructure-as-a- service* for testers, tools, applications, etc., in order to form the backbone for the different types of tests that are part of mobile TaaS. Our system MTaaS could form an integral piece of mobile TaaS on which application services such as testing tools and methods can be based on. The request generator of MTaaS will be the basis for generation of diverse tests and the resource allocation and provisioning via hybrid intelligent algorithms. They will be used to automatically allocate resources per demand basis on the MTaaS system. A comparison to the existing major players is presented in Table 10. The main difference between existing players and MTaaS is that we developed infrastructure service for mobile TaaS. In addition, we adopt device cloud and emulation cloud as the major infrastructure approach. MTaaS is a more accessible option compared to the current market offerings. Most of the current systems require that users join a community and network to find human testers in order to test applications.

Our service aims to automate the median between the developer and the tester to provide access to the smart phones and the resources to test an application, in order to eliminate the need for a network of human testers due to our automation of black box tests that we are developing.

In addition, we aim to perform testing on actual cell phones unlike many other mobile application testing services depending on emulators to generate test results. Applications may work well on emulators but then may crash when tested on an actual device. In order to quickly send out multiple smart phone devices for testing, our platform and network of smartphones will be readily available at any time. Since the operations are scalable without incurring a large cost, MTaaS can meet virtually any amount of demands for service. If the demand for the service ever did rise beyond what our current test devices were able to process, we could

Table 10
A comparison to the existing major players.

Major players	Testin	Yiceyun	PerfectoMobile	uTest	TestDroid	Our system (MTaaS)
Infrastructure approach	Crowdsourcing	Crowdsourcing	Emulation cloud	Crowdsourcing	Emulation cloud	Device cloud, emulation cloud, support crowdsourcing users
Real devices	yes, 4000 +	Yes, 200 +	Yes, 200 +	no	Yes, 400 +	Yes, by users, providers, and donated reources
Auto-test	No	No	Yes	No	Yes	Yes
Deploy method	Central control plus user device	User device	Continuous quality Lab	User device	Centers in US and Europe	Device cloud
OS type	iOS, android	Android	iOS, android	iOS, android	iOS, android	iOS, android
Testing focuses	Install/uninstall, compatibility, function, performance	Adaption, traverse, performance	Remote, performance, monitoring	Global testing community	Function, performance	On-demand, scalable testing,
Service type	PaaS	PaaS	PaaS	PaaS	PaaS	IaaS
Test community	Yes	No	No	Yes	No	No
24/7/365 service	Yes	No	No	No	No	Yes
Security testing	No	No	No	Yes	No	No

offer for more smartphone owners to earn money by turning their old phones into test devices as resource donation.

Moreover, MTaaS supports crowdsourcing mobile cloud based testing. Through the platform, clients are able to remotely connect app developers to devices (smartphones) that are located virtually anywhere in the world as long as the devices are powered on and connected to the platform via the internet. Though this type of crowdsourcing testing already exists in some way, they require a network of people to manually run the tests on the devices that are connected to the platform. Besides IaaS, we are currently developing test automation approaches in order to reduce the demand for a knowledgeable human tester to manually run tests on the device. Along with that, we are developing an app that anyone with a smartphone can download that will connect their phone to our platform as a test device and become part of our network. Once the devices are connected, the developers can remotely install their app on any number of devices in our network, and run automated tests. Once the tests are concluded, our platform will uninstall the app, and then send a bug report back to the app developer so they can determine what they need to do in order to get their app working properly on that particular device. Throughout this process, the owner of the device does not have to do anything other than leave it on and connected to the internet with our app running.

Automating the human element of mobile testing (as well as purchasing a warehouse of phones) results in significant cost savings. MTaaS aims to solve the problem of smaller app developers not having sufficient access to a variety of different physical test devices to test their new apps on. Additionally, our service would be an alternative for bigger app developers to purchasing and running their own testing lab, and as a way to outsource their mobile testing needs for cost savings.

4.6. The limitations of current MTaaS

The current MTaaS system suffers from several limitations as follows.

The developed system in this paper only focuses on IaaS level, i.e., on-demand infrastructure service of test resources (device, emulators, hubs, TaaS servers) for mobile app testing. To provide a complete mobile TaaS system, PaaS-level service need to be developed. Currently, we are working on test automation approaches for mobile apps, including test environment setting, test models, and test methods.

Regarding security issue, user access control is considered in the system. Nevertheless, mobile resource security needs to be enhanced in the future. As the initial version of mobile IaaS, we primarily focus on features such as service infrastructure, virtualization solution, implemented functions and system performance. In market applications, cloud services encounter a number of the risks associated with security and privacy. There are several security issues that need to be addressed, such as the security and privacy of user's data stored on cloud server(s), security threats caused by multiple virtual machines, and intrusion detection. We plan to address some security issues in the future work. More issues and challenges in mobile TaaS will be discussed in the next section.

5. Issues, challenges, and needs in mobile TaaS

Although there are a number of published papers addressing issues, challenges and needs in testing services (Gao et al., 2011; Gao et al., 2014) and mobile testing (Gao et al., 2013; Gao et al., 2012), no publications discusses the challenges and needs in mobile testing-as-a-service on cloud. In previous work, we addressed some testing issues, such as testing models, testing automation approaches in mobile testing and mobile TaaS. However, there are more challenges existed in testing criteria and standards, large-scale test script automation and cloud-based Mobile TaaS. This section summarized several current issues.

Issue #1: *Lack of well-defined infrastructures and approaches on cloud which allow both mobile application vendors and users to access for mobile TaaS services.*

Since cloud-based mobile TaaS is a new topic among software testing, mobile computing, and software testing service communities, now we are at the earlier stage. As more people recognize the importance of its business opportunities and needs, more mobile device clouds (such as sensor device clouds and mobile device test clouds) will be developed to support mobile TaaS cloud infrastructures and services. In addition, with the population of crowd-based application, how to construct effective service infrastructure for crowd-based mobile users and crowed-souring server is still a problem.

Need #1: *More study and research results on cloud-based mobile TaaS infrastructures and mobile testing environments supporting on-demand elastic mobile testing resources and offering unified mobile test automation services.*

Since mobile scalability is one important QoS evaluation parameter, engineers must validate it using both emulation-based and

device-based testing approaches. To support this, engineers need to an elastic cloud-based mobile test infrastructure based on a remote mobile device cloud and mobile virtualization. Hence, a desirable mobile TaaS infrastructure is needed to support auto-provision of mobile testing resources with elastic scalability.

Issue #2: *Lack of standards in mobile test environments, billing charges, tools, and test automation for mobile testing services on cloud.*

As pointed out in Gao et al. (2012), engineers have found this problem in mobile test automation even though there are numerous available mobile test tools since they used different interfaces, scripting languages, and diverse technologies. Thus, setting up a mobile test environment for multiple applications across various platforms is tedious, time-consuming, and expensive. Moreover, the frequent updates and changes in mobile device and platforms worsen this situation.

Need #2: *Developing well-defined standards for mobile TaaS service on cloud in different areas.*

Several types of standards are needed. The first is a standard cloud-based mobile test environment which can be easily defined, configured, deployed, and executed. And the other refers to standard testing service protocols and APIs supporting test services, which includes the followings.

- Standard TaaS interactions between mobile device clouds and the mobile TaaS server.
- Standard reusable mobile test platform with well-defined common testing services on mobile devices.
- Standard tool control interactions/interfaces between mobile test tools and under-test mobile applications.
- A standard mobile connectivity control service protocol for mobile device clouds to support and control mobile connection for each mobile device.

Issue #3*: There is a lack of well-defined test models and coverage criteria to address distinct needs in cloud-based mobile testing.*

The existing test models did not address and present diverse mobile environment contexts (such as mobile platforms, web browsers, mobile technologies, different native APIs, and device-specific gesture, and related configurations on different devices), diverse network connectivity and related contexts, testing scalability and mobility, usability and security. These problems need to be solved in cloud-based mobile TaaS.

Need #3: *Well-defined test models and criteria to address the special features in mobile applications.*

Recently, several researchers attempt to address this need in mobile testing. For example, a semantic tree model is proposed in Tao and Gao (2014) to present diverse testing environments in mobile testing. The approach in Aktouf et al. (2015) aims to test location-based function service using a dynamic graph-based model. However, there is still a demand for more test models and coverage criteria to address the special features, such as mobility, cross-platform, and crowdsourcing.

6. Conclusions

This paper presents the design and development of a mobile TaaS system known as MTaaS as *infrastructure-as-a-service*. The system supports mobile TaaS by providing mobile instances such as server machines, mobile hubs, devices and emulators for testing environment and applications. The evaluation results indicate the feasibility and effectiveness of the system. In addition, the paper provides the detailed perspectives on the current mobile TaaS, including the test features, infrastructures, industrial practices, as well as issues and challenges. Moreover, a comparative study is given to indicate the advantage of the developed system.

As the fast increase of mobile app deployments on devices, engineers need more quality validation research and test automation tools to cope with the discussed issues and challenges. Mobile TaaS is an effective approach to on-demand and large-scale mobile testing through providing services like infrastructures, platforms, and applications. Mobile test automation solutions are also urgently needed to meet current and future demands in mobile testing. Besides the developed MTaaS system for mobile TaaS, we are currently developing a test automation platform as *platform-as-a-service* based on IaaS as another important piece of mobile TaaS.

Acknowledgment

This paper is supported by the National Natural Science Foundation of China under Grant No. 61402229 and No. 61502233; the Open Fund of the State Key Laboratory for Novel Software Technology (KFKT2015B10), and the Postdoctoral Fund of Jiangsu Province under Grant No.1401043B.

References

Aktouf, O.E.K., Zhang, T., Gao, J., Uehara, T., 2015. Testing location-based function services for mobile applications. In: Accepted in Proceedings of The First International Workshop on Mobile Cloud TaaS (MCTaaS 2015).
Amalfitano, D., et al., 2012. Using GUI ripping for automated testing of android applications. In: Proceedings of the IEEE/ACM International Conference on Automated Software Engineering, pp. 258–261.
Anand, S., et al., 2012. Automated concolic testing of smartphone apps. In: Proceedings of the ACM SIGSOFT International Symposium on the Foundations of Software Engineering, pp. 1–11.
Bai, X.Y., Li, M.Y., Tsai, W.T., Gao, J., 2013. Vee@Cloud: the virtual test lab on the cloud. In: Proceedings of the Workshop on Automation of Software Test (AST), pp. 15–18.
Bo, J., Xiang, L., Gao, X.P., 2007. MobileTest: a tool supporting automatic black box test for software on smart mobile devices. In: Proceedings of the International Workshop on Automation of Software Test.
Buyya, R., et al., 2009. Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: challenges and opportunities. In: Proceedings of International Conference on the High Performance Computing & Simulation (HPCS), pp. 1–11.
Do, T.V., Rotter, C., 2012. Comparison of scheduling schemes for on-demand IaaS requests. J. Syst. Softw. 85, 1400–1408.
Dorgio, M., Birattari, M., 2010. Ant colony optimization. In: Encyclopedia of Machine learning. Springer, US, pp. 36–39.
Gao, J., et al., 2012. A cloud-based TaaS infrastructure with tools for SaaS validation, performance and scalability evaluation. In: Proceedings of IEEE International Conference on Cloud Computing Technology and Science (CloudCom), pp. 464–471.
Gao, J., et al., 2013. Mobile application testing: a tutorial. IEEE Comput. 47 (2), 46–55.
Gao, J., Bai, X.Y., Tsai, W.T., Uhere, T., 2011. Cloud testing - issues, challenges, needs and practice", software engineering. Int. J. (SEIJ) 1 (1), 9–22.
Gao, J., Tsai, W.T., Paul, R., Bai, X.Y., 2014. Mobile testing-as-a- service (mobile taas)--infrastructures, issues, solutions and needs. In: Proceedings of IEEE International Symposium on High Assurance Systems Engineering (HASE),, pp. 158–167.
Hargassner, W., et al., 2008. A script-based testbed for mobile software frameworks. In: Proceedings of International Conference on Software Testing, Verification, and Validation, pp. 448–457.
Hirenkumar, B.H., 2015. An overview of load balancing techniques in cloud computing environments. Int. J. Comput. Appl. 4 (1), 9874–9881.
Mann, Z.A., 2015. Allocation of virtual machines in cloud data centers—a survey of problem models and optimization algorithms. ACM Comput. Surv. 48 (1), 1–34 Article 11.
Manvi, S.S., Shyam, G.K., 2014. Resource management for infrastructure as a service (IaaS) in cloud computing: a survey. J. Netw. Comput. Appl. 41, 424–440.
Muccini, H., Francesco, A.D., Esposito, P., 2012. Software testing of mobile applications: challenges and future research directions. In: Proceedings of International Workshop on Automatic Software Test Automation, pp. 29–35.
Openstack documentation. http://docs.openstack.org/.
Ridene, Y., Barbier, F., 2011. A model-driven approach for automating mobile applications testing. In: Proceedings of the 5th European Conference on Software Architecture, pp. 1–7.
Riungu, L.M., Taipale, O., Smolander, K., 2010. Software testing as an online Service: observations from practice. In: Proceedings of International Conference on Software Testing, Verification, and Validation Workshops (ICSTW), pp. 418–423.
Satoh, I., 2004. Software testing for wireless mobile computing. IEEE Wireless Commun. 11 (5), 58–64.
Satoh, I., 2012. A testing framework for mobile computing software. IEEE Trans. Softw. Eng. 29 (12), 1112–1121.

Tao, C.Q., Gao, J., 2014. Modeling mobile application test platform and environment: testing criteria and complexity analysis. In: Proceedings of the 2014 Workshop on Joining AcadeMiA and Industry Contributions to Test Automation and Model-Based Testing, pp. 28–33.

Tsai, W.T., Hang, Y., Shao, Q.H., 2011. Testing the scalability of SaaS applications. In: Proceedings of IEEE International Conference on Service-Oriented Computing and Applications (SOCA), pp. 1–4.

Yang, Y., Onita, C., Zhang, X., Dhaliwal, J., 2010. TESTQUAL: conceptualizing software testing as a service. e-Service J. 7 (2), 46–65.

Yehuda, OA.B, Yehuda, M.B., Schuster, A., Dan, T., 2014. The rise of RaaS: the resourceas-as-a-Service cloud. Commun. ACM 57 (7), 76–84.

Zhan, Z.H., Liu, X.F., Gong, Y.J., Zhang, J., Chung, H.S., Li, Y., 2015. Cloud computing resource scheduling and a survey of its evolutionary approaches. 47 (4), Article 63:1–33.

Chuanqi Tao is an assistant professor in the Department of Software Engineering at Nanjing University of Science and Technolgy in Nanjing, China. His research areas include model-based testing and test aumomation, mobile application testing, and regression testing. Contact him at taochuanqi@njust.edu.cn.

Jerry Gao is a full professor in the Department of Computer Engineering at San Jose State University, CA, USA. His research areas include cloud computing, testing-as-a-service (TaaS), mobile computing, and software testing and automation. Contact him at jerry.gao@sjsu.edu.