MOBILE APPLICATION WITH CLOUD COMPUTING

V.L.DIVYA

M.E, COMPUTER SCIENCE AND ENGINEERING
ANAND INSTITUTE OF HIGHER TECHNOLOGY
CHENNAI, TAMIL NADU
divibiju07@gmail.com

Abstract- Smartphones enable a new, rich user experience in pervasive computing. The major problem with Smartphone is that hardware resources such as CPUs, memory and batteries are still limited. To solve this resource problem, many researchers have proposed architectures to use server resources in the cloud for mobile devices. This paper proposes a conceptual architecture where mobile application platform share the software as a service among multiple users on cloud server via network.

Index Terms- Android, Multi-tenant, Cloud, Virtual Smartphone

I. INTRODUCTION

The architecture for remotely using mobile application on server is called Mobile Application Platform on Cloud Server that intends to handle not only user data but also user applications in a cloud server. This approach changes the application lifecycle as follows. "Write once, run everywhere. Install once, use everywhere."Android is an open-source mobile OS initiated by Google. The main reason to select Android as a server platform is that it is able to run not only for Smartphone but also for x86 processor.

Software as a Service (SaaS) represents a novel paradigm and business model expressing the fact that companies do not have to purchase and maintain their own ICT infrastructure, but instead, acquire the services embodied by software from a third party. Here SaaS service is provided for mobile users. This paper proposes Android as a Server Platform that enables many users to use resources on remote cloud servers. This paper also proposes a multi-tenant architecture of Android on cloud server.

Multi-tenancy, this is defined as a feature where the software running on a server provides services to many users. It is one of the important features for cloud computing. From the viewpoint of both economy and ecology, it is beneficial to share hardware resources among users. Using a mobile OS would be more effective than using a desktop OS because the resource requirements of mobile OSs are smaller.

II. MULTI-TENANCY

Introduction

Multi-tenancy is an organizational approach for SaaS applications. Although SaaS is primarily perceived as a busi - ness model, its introduction has lead to numerous interesting problems and research in software engineering. Despite the growing body of research in this area, multi-tenancy is still relatively unexplored, despite the fact the concept of multi-tenancy first came to light around 2005.

The various definitions focus on what is believed to be the key aspects of multi-tenancy:

a) The ability of the application to share hardware resources.
b) The availability of a high degree of configuration of the software.
c) The architectural approach in which the tenants (or users) make use of a single application and database instance.

Multi-tenant Versus Multi-User

It is necessary to make an important, but subtle distinction between the concepts multi-tenant and multi-user. In a multi-user application it is assumed that all users are using the same application with limited configuration options. In a multi-tenant application, it is assumed that each tenant has the possibility to heavily configure the application. This results in the situation that, although tenants are using the same building blocks in their configuration, the appearance or workflow of the application may be different for two tenants.

Architecture

Multi-tenancy is achieved through different approaches. The first approach is by running multiple users Virtual Machines in a server via a hypervisor. This approach has the advantage of application usability and maintenance. From the viewpoint of application usability, every mobile application that can run on Android-x86 is usable because each Android OS runs only one application.

The second approach implements multi-tenant function in kernel-layer. This approach changes Android OS to run multiple user applications in separate processes. This approach is similar to an ordinary thin client server running multiple user applications in a server. The main challenge is that original Android supports only one display and keypad device since Android is mainly designed to work on a Smartphone.

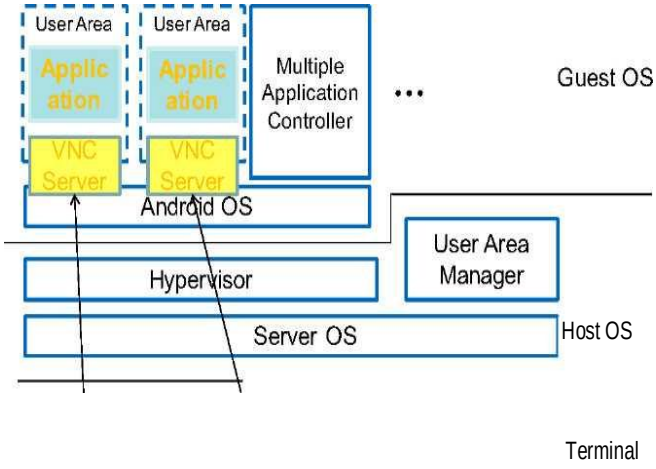
Another approach is to create a multi-tenant function at framework-layer, similar to existing Java-based multi-tenant framework. This approach remodels Android the framework and

APIs to support multiple user applications. The main challenge is how to run existing Android applications in modified framework.

Implementation

In the functional overview of the architecture two new functions are defined for enabling multi-tenant for Android. The first function is the multiple application controller installed in an Android OS, and the second is the user area manager located in a host OS. The multiple application controller enables running of multiple applications as if each application is running on independent physical Smartphone. It is important requirement to decrease implementation cost for Android OS because of maintenance about OS version up problem. The user area manager controls server resources and act as an interface between a terminal and the multiple application controllers.

When user wants to use an application, the user terminal contacts the user area manager and order to launch application. The user area manager checks the server resources and select which guest OS to run application. The multiple application controller launches the application based on an order from the user area manager. The user area manager returns VNC connection information such as IP address and port.



.

Fig.1: Functional Overview of Multi-Tenancy for Android

Key Characteristics of Multi-Tenancy

Hardware Resource Sharing: The concept of multi-tenancy comes in different flavours, and depending on which flavour is implemented, the utilization rate of the underlying hardware can be maximized.

The following variants of multi-tenancy can be distinguished

- Shared application, separate database.
- Shared application, shared database, separate table.
- Shared application, shared table (pure multi-tenancy)

High Degree of Configurability: In a single-tenant environment, every tenant has his own, (possibly) customized application instance. In contrast, in a multi-tenant setup, all tenants share the same application instance, although it must appear to them as if they are using a dedicated one. Because of this, a key requirement of multi-tenant applications is the possibility to configure and/or customize the application to a tenant's need, just like in single-tenancy .

Shared Application and Database Instance: A single-tenant application may have many running instances and they may all be different from each other because of customization. In multi-tenancy, these divergences no longer exist as the application is runtime configurable. This entails that in multi-tenancy the overall number of instances will clearly be much lower (ideally it will be one, but the application may be replicated for scalability purposes). As a consequence, deployment is much easier and cheaper.

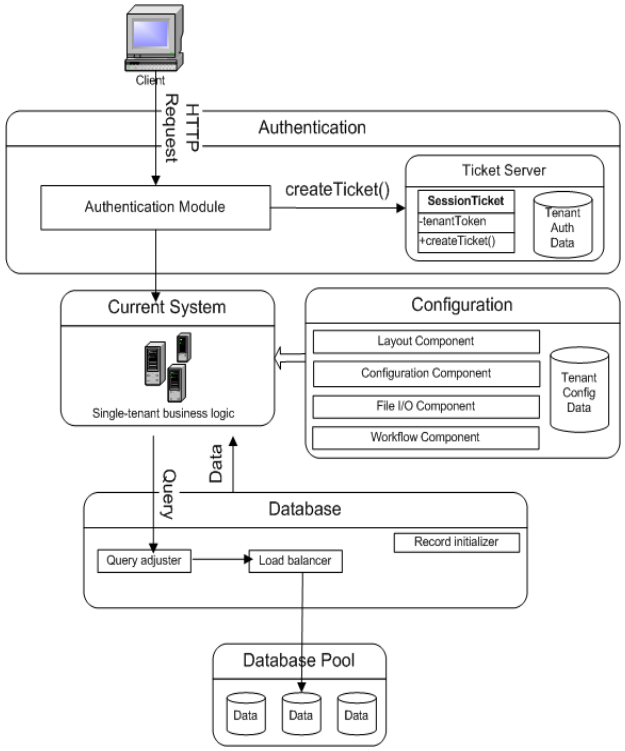


Fig 2: Architectural Overview of Multi-tenancy

III. USING MOBILE DEVICES AS A VIRTUAL CLOUD COMPUTING PROVIDER

Motivation and Scenario

While considering the case of offloading to devices with similar characteristics, in which the performance will be similar to the source node, the overall performance of the task will be worse than running it on a single device due to the migration overhead. Therefore it is needed to explore what makes the offloading to similar devices beneficial.

On an economical basis, accessing cloud computing providers is associated with two costs: the cost of networking plus the cost of using the provider's resources. The latter is not high nowadays – it can be as cheap as 5 USD per month considering the access of a small on-demand server 2 hours per day 2 - but is expected to increase to reach higher levels of uptime and better support .

On the other hand, wireless data fee is still very high. As an example, in South Korea the subscription plans for the i-phone 3GS (32GB) are near 70 USD per month, and if the user wants to download 1 GB of data he has to pay more than 200 USD. Besides, using 3G connectivity consumes more battery and is slower than network interactions with nearby devices using other interface such as WiFi.

On a technical side, there are several benefits to consider: First, it is necessary to preserve conventional offloading benefits, such as allowing applications that cannot, otherwise, be executed on mobile devices due to a lack of resources. For example, if memory is not enough then creating instances of those objects on any remote device will allow the application to be executed. Second, performance can be enhanced if the execution sequence of an application can be reordered for increasing the level of parallelism.

Architecture

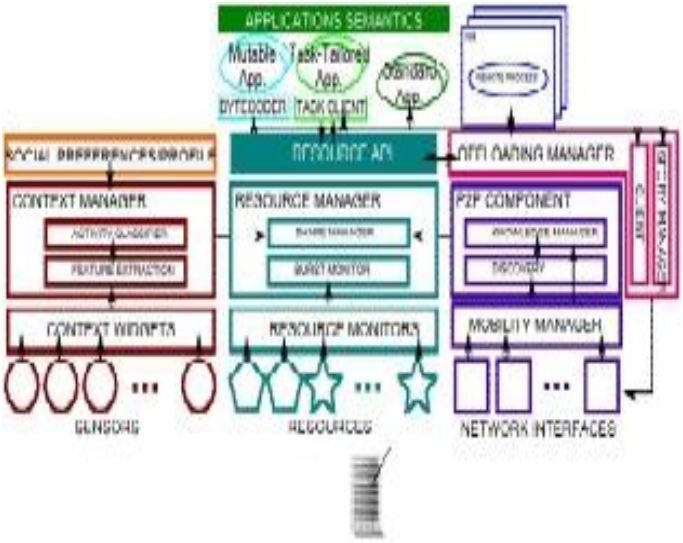


Fig 3: Architecture of Ad Hoc Mobile cloud

The process for the creation and usage of a virtual cloud provider is simple: If a user is at a stable place and wants to execute a task which needs more resources than available at the device, the system listens for nodes in the vicinity. If available, the system intercepts the application loading and modifies the application in order to use the virtual cloud. To support this process, the architecture proposed consists of five main features: Application manager; Resource manager; Context manager; P2P component and; Offloading manager.

Application Manager: It is in charge of launching and intercepting an application at loading time and modifying and application to add features required for offloading – proxy creation, RPC support - according to the current context. Since the idea is to replace calls to infrastructure-based clouds, the interception and modification should focus on modifying the reference to that provider with a reference to the virtual provider. This process is performed when an application is executed the First time. Once an application is modified, its modified copy is used to avoid further delays.

The Resource Manager: It is in charge of application profiling and resource monitoring on a local device. For each application, a profile is defined in terms of the number of remote devices needed to create a virtual cloud, and sensibility to privacy and amount of resources needed for the migration to happen (in average). This profile is checked by the application manager whenever an application is executed in order to determine whether an instance of the virtual provider should be created or not.

The Context Manager: It wields and synchronizes contextual information from context widgets and makes it available in some way for other processes. It is composed by three subcomponents: context widgets that communicate with the sources of information; a context manager itself that handles the information and extracts new contexts from them; and a social manager that is used to store the knowledge regarding relationship between users.

P2p Component: Two basic contexts are of utter importance are the location and number of nearby devices. The former is used for the mobility traces. The later for the enabling of a cloud from the application manager, and it is given by the P2P component. This component is aware of the status of the devices in the surroundings: it sends events to the context manager in case a new device enters the space, or if an existing device leaves the space. It utilizes an ad hoc discovery mechanism, and then groups the nodes using a P2P scheme, allowing for better scalability and distribution of contents.
Once that information is captured, a context aggregator located in the context manager generates high level contexts from the basic contexts. They represent the consolidated information related to the user. It is only defined with one high level context for this framework, which is whether the user is in a stable location or not.

Offloading manager component: It is in charge of sending and managing jobs from the node to other remote devices, plus receiving and processing jobs sent from them. It communicates with the P2P component once a job is issued to the respective node, and waits for the results to be delivered back to the application. This component is the one in charge of detecting failures in the execution and to re-emit them. It also is in charge of creating protected spaces for the execution of the tasks coming from remote nodes. This protected spaces (represented here as a VM), are utilized to block the access to sensitive data located on the devices

Implementation

A prototype is implemented with framework in Java. It was selected because it provides all the needed capabilities in terms of intercepting the loading, modifying the classes and also there were implementations available for cloud computing providers and clients on top of this platform. A main issue was to modify applications in order to intercept and replace references to infrastructure-based clouds with mobile ones. In Java, code

interception can be done using bytecoders in conjunction with a personalized class loader component, built on top of regular JVM loader. A bytecode generator creates and injects the needed code, while the loader allows the interception of the classes before loading them in memory. A personalized version of Javassist[14] was utilized for this purpose.

Communication between devices is based on the Extensible Messaging and Presence Protocol (XMPP). Yaja! I is modified as a XMPP client is implemented in Java. The modifications allow us to be able to execute Yaja! on mobile devices and to incorporate two extensions for XMPP: Serverless Messaging8 and Jabber RPC9. The former is based on mDNS and ZeroConf, and allows for the discovery and messaging among devices without the need of an infrastructure. The latter is a scheme based on XML-RPC using XMPP as the transport protocol, and it is utilized to execute the remote tasks associated with a cloud job.

IV. VIRTUAL SMARTPHONE OVER IP

Virtual Smartphone over IP, which provides cloud computing environment specifically tailored for smartphone users. It allows users to create virtual smartphone images in the cloud and to remotely run their mobile applications in these images as they would locally. The motivation is to allow smartphone users to more easily tap into the power of the cloud and to free themselves from the limit of processing power, memory and battery life of a physical smartphone. Using the system, smartphone users can choose to install their mobile applications either locally or in the cloud.

Running applications remotely in the cloud has a number of advantages, such as avoiding untrusted applications from accessing local data, boosting computing resources, continuing to run applications on the background and opening up new way to use smartphones.

Virtual Smartphone over IP system adopts architecture similar to ones commonly used by server hosting providers. As illustrated, the system is composed of a number of external smartphone clients, a front-end server, a virtual smartphone farm, a management server and a network file system (NFS).

- Virtual smartphone farm is the most important component of the system. It is a virtualization environment that hosts a collection of virtual smartphone images, each of which is dedicated to a smartphone user.

- The front-end server admits service requests from smartphone users across the Internet and establishes remote sessions to the appropriate virtual smartphone images. The frond-end server also allows smartphone users to create, configure and destroy virtual smartphone images. Once a remote session is established, the user can install and run mobile applications on one of these images instead own physical smartphone.

- The network file system is used by virtual smartphones for all persistent file storage, in much the same way that an SD card holds data for physical smartphones. Since the NFS is easily scalable, it practically provides each virtual smartphone unlimited file storage.

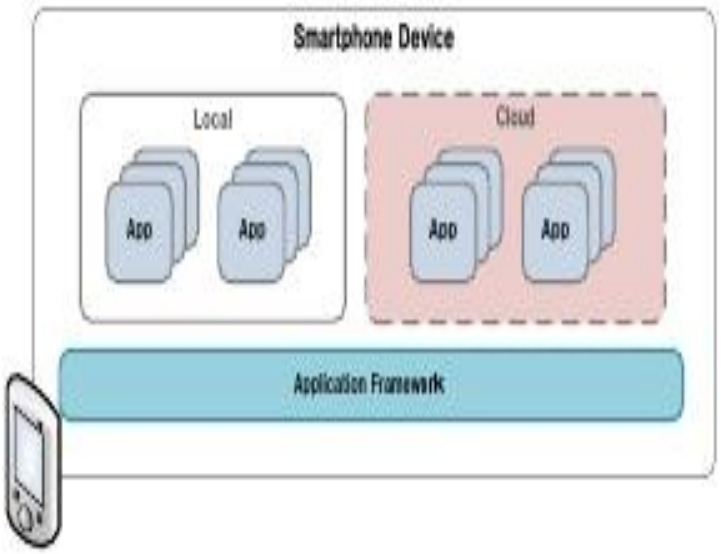


Fig 4: Basic concept of System

- The management server is used to manage the virtual smartphone farm. Typical operations of a management server include the creation of virtual images in bulk and troubleshooting individual images.

Users control their virtual smartphone images through a dedicated client application installed on their smartphones. This client application receives the screen output of a virtual smartphone image and presents the screen locally in the same way as conventional thin-client technology. Since it is expect that most users to access their virtual smartphone images through an unstable network such as 3G, the image must continue to run on the farm and be in the same state when the user is connected again after the user is disconnected in an expected manner.

Implementation

A proof-of-concept prototype is proposed using Android, an open-source mobile OS initiated by Google. The main reason behind the choice is that Android OS is not only designed for smartphone devices with an ARM processor, but also is being ported to the x86 platform [4]. Although Android-x86 is originally intended for netbooks, it gives us an opportunity to create a virtual image of Android using a bare-metal hypervisor. This allows each virtual Android-x86 image to tap into the power of server hardware in a data center. The fact that a CPU emulator is not needed (i.e. x86-to-ARM) to run the virtual image is very important since such emulator always introduces enormous overhead and may neutralize any performance advantage offered by a data center.

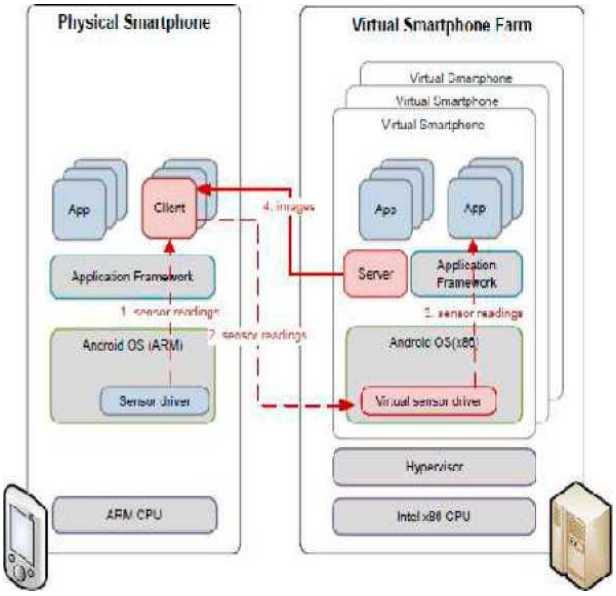


Fig 5: Overview of Virtual Smartphone over IP

A client application can be implemented on an Android smartphone. Although the system does not require the physical and the virtual smartphones to be on the same platform, this particular setting allows us to more tightly integrate both environments.

A pair of VNC-based server can be implemented on client programs. The server program resides in each Android-x86 image that run on top of VMWARE ESXi while the client program is installed in the physical Android device. The client program enables a user to remotely interact and control Anroid-x86 images. The client program transmits various events from the physical device to the virtual smartphone and receives graphical screen updates from the virtual smartphone.

A virtual sensor driver can also be implemented in the Android-x86 image. Most modern smartphones are equipped with various sensor devices such as GPS, accelerometer and thermometers. While VNC itself supports only keyboard and mouse as the primarily input devices, it can also be extended where client program can transmit sensor readings (accelerometer, orientation, magnetic field and temperature etc) to the virtual sensor driver in the Android-x86 image. The virtual sensor driver is implemented in such a way that the sensor readings from the physical Android device would appear to come from the Anroid-x86 images itself. This is an important feature as it allows Android applications in an Android-x86 image to obtain sensor readings from the physical smartphone without any modification.

Proposed prototype allows applications running in the cloud to appear like local applications on the physical device, with functions such as copy-and-paste between local and remote applications. The prototype also features remote shortcuts to remote applications in the virtual smartphone that minimize the number of steps required for users to launch remote applications. Furthermore, each short-cut can point to a different virtual Android-x86 image, and thus allowing users instant access to remote applications residing in multiple Android-x86 images in one single menu.

V. ANDROID OS

The reasons where run Android OS on hypervisor is implemented are as follows:

- *Maintenance:* An OS image is easy to backup, restore, and check the server environment similar to other hypervisor-based virtualization.

- *Application **E**nvironment:* The Android OS has backward compatibility regarding application runtime. For example, an application that is made for Android 1.6 runtime can work in Android 2.1 runtime. However, device information such as sensor, keyboard, and display size, varies from one physical device to another. Running Android OS in various setting is simple method to keep application environment.

- *CP**U** management:*

The Android is not designed to run an a multiple core environment. Therefore hypervisor's CPU allocation is important to strengthen the efficiency of CPU power.

Multiple application controllers is an enabler to run multi user applications. The discussion is focused about device support for multi user and how to run application.

Multiple device support for Android Application: From the viewpoint of the Android application lifecycle, only one application can use the display, and the other applications automatically run in the background process. To solve this issue, multiple display device support and application mapping is required.

*Data Security Integration**:*** The Android has security mechanism to protect the application data sector from other application using user account based access control. Of course Android is not designed to use same application for different users. So, if users share same application data sector, mixing of unexpected user data may occur.

There are two approaches to solve this problem. One is virtual SD card approach. When all user accesses to SD card is controlled, it can prevent access to another user's application data. The benefit of this approach is that implementation impact is limited to SD card driver. Second approach is using a file system function. GNU chroot enables changing the application root directory dynamically. This approach would also prevent unexpected file access.

VI. CONCLUSION

This paper, proposes Android as a Server Platform system that enables the use of sharing server-side Android OS among multiple users. It is possible to develop a prototype system about

proposed multi-tenant Android architecture. It is believed that proposed architecture shows high performance on virtual image-based virtualization for mobile applications.

ACKNOWLEDGMENT

I am really thankful to the Almighty. I am also thankful to my guide Mrs. Roselin Mary, HOD of Computer Science Department, AIHT, who helped me for various researches in the project. I also convey my thanks to all the staff members of AIHT for helping me out in this project.

REFERENCES

[1] M. Satyanarayanan, V.Bahl, R. Caceres, and N. Davies, The Case for VM-based Cloudlets in Mobile Computing. : IEEE Pervasive Computing, 2009.

[2] G.H-Canepa and D.Lee A Virtual Cloud Computiong Privoder for Mobile Devices. San Francisco : MCS'10, 2010.

[3] B.G. Chun and P. Maniatis. Augmented Smartphone Applications Through Clone Cloud Execution.

[4] Y.Royon, S.Frenot, and F.L.Mouel Virtualization of Service Gateways in Multi-provider Environments. Heidelberg : CBSE 2006, 2006.

[5] C.P.Bezemer and A.Zaidman Multi-Tenant SaaS Applications: Maintenance Dream or Nightmare? Antwerp, Belgium : IWPSE-EVOL' 10, 2010.

[6] E.Y.chen and M.Ito. Virtual Smartphone over IP. Montreal,QC, Canada: IEEE WOWMOM, 2010.

[7] Android-x86 Project - Run Android on Your PC(Android-x86 - Porting Android to x86). http://www.android-x86.org/.

[8] Application Fundamentals | Android Developers. http://developer.android.com/guide/topics/fundamentals.html