MOBILE CLOUD

1.3 MOBILE CLOUD

Limited resources is another critical characteristic of mobile devices. With the development of cloud computing, mobile cloud computing has been introduced to the public. Mobile cloud computing, as shown in Fig. 1.5, is the combination of cloud computing, mobile computing, and wireless networks to bring rich computational resources to the mobile system. In general, a mobile device with limited resources can utilize computational resources of various cloud resources to enhance the computational ability of itself. There are several challenges in mobile cloud computing, such as moving computational processes from mobile devices to the cloud, networking latency, context processing, energy management, security, and privacy.

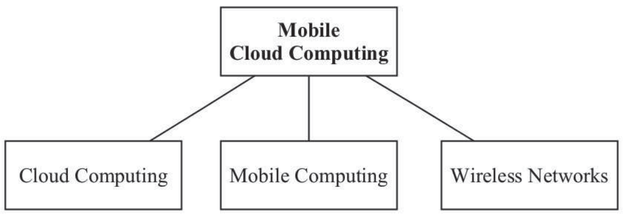

Figure 1.5 Main structure of mobile cloud computing.

Currently, some research and development addresses execution code offloading, seamless connectivity and networking latency; however, ef- forts still lack in other domains.

Architecture. The architecture for a heterogeneous mobile cloud computing environment is crucial for unleashing the power of mobile computing toward unrestricted ubiquitous computing.

Energy-aware transmission. Offloading executive codes into the cloud can greatly reduce the burden and the time of local mobile devices, but increase the transmission between mobile de- vices and the cloud. The transmission protocol should be carefully designed for saving energy.

Context-aware computing. Context-aware and socially aware computing are inseparable traits of mobile devices. How to achieve the vision of mobile computing among heterogeneous con- verged networks among mobile devices is an essential need.

Live Virtual Machine (VM) migration. A virtual machine is an emulation of a particular computer system. Executive re- source offloading involves encapsulation of a mobile app in a VM instance, and migrating in the cloud is a challenging task.

Security and privacy. Due to lack of confidence in the cloud, many users are concerned with the security and privacy of their information. It is extremely important to improve the security and the privacy of mobile cloud computing.

1.3.1 Big Data Application in Mobile Systems

Big data is an all-encompassing term for any collection of data sets so large or complex that it becomes difficult to process them using traditional data processing applications. Data sets grow in size in part because they are increasingly being gathered by mobile devices. There are 4.6 billion mobile phone subscriptions worldwide and between 1 billion and 2 billion people accessing the Internet.

With billions of mobile devices in the world today, mobile computing is becoming the universal computational platform of the world. These mobile devices generate huge amounts of data every day. The rise of big data demands that we be able to access data resources any- time and anywhere about every daily thing. Furthermore, these kinds of data are invaluable and profitable if used well.

However, a few challenges must be addressed to make big data analytics possible. More specifically, instead of being restricted to single computers, ubiquitous applications must be able to execute on an ecosystem of networked devices, each of which may join or leave the shared ubiquitous space at any time. Moreover, there exist analytics tasks that are too computationally expensive to be performed on a mobile device ecosystem. Also, how can we harness the specific capabilities of each device, including varying display size, input modality, and computational resources?

1.3.2 Data Security and Privacy Protection in Mobile Systems

Due to the universality and the particularity of mobile systems to desk- top system, the security in mobile systems is much more complicated and important than that in desktop systems. The security in mobile systems can be separated into a few parts.

The first threat is the malware (virus). Mobile malware is a malicious software that targets mobile devices and results in the collapse of the system and loss or leakage of information. According to the June 2014 McAfee Labs Threat Report, new mobile malware has in- creased for five straight quarters, with a total mobile malware growth of 167 percent in the recent past years. Security threats are also growing with 200 new threats every minute. In addition to 2.4 mil- lion new samples of mobile malware, 2013 also brought 1 million new unique samples of ransomware, 5.7 million new malicious signed binaries, and 2.2 million new Master Boot Record (MBR)-attack-related samples. The most frequent two incentives are exfiltrating user information and premium calls or SMS. Furthermore, there are some other incentives, such as sending advertisement spam, novelty and amusement, and exfiltrating user credentials.

Another research issue is the security frameworks or approaches for detecting mobile malware. There are several approaches for monitoring mobile devices and detecting mobile malware. The signature- based solution is an approach used for detecting attacks, but it fails miserably in detecting the sophisticated cyber-criminal who targets specific organizations with exploits tailored to those victims. From a process perspective, when it comes to validating a threat and subsequent root cause analysis, first-level responders have to send all data that looks like malicious code to the reverse engineers. This process often causes delays, because these malware teams are typically inundated.

Meanwhile, with the development of technology, an efficient repre- sentation of malware behaviors using a key observation often reveals the malicious intent even when each action alone may appear harm- less. The logical ordering of an application's actions are often over time. Based on this idea, researchers present various approaches to monitor and detect malicious behavior using static analysis on data flow.

Next security problem is the data over-collection behaviors in mobile apps. Current mobile phone operating systems only provide coarsegrained permissions that determine whether an app can access private information, while providing few insights into the scope of private in- formation being used. Meanwhile, only a few users are aware of per- missions information during the installations. Furthermore, some users choose to stop installing or to uninstall an app when the system warns them and asks for permission, even though they know it may bring some hidden security troubles. For example, we take location data and analyze the current status and discuss the risks caused by over collect- ing it.

Location data are the most frequently used data in smartphones. It can be used in apps whose main functions include maps, photo or- ganization, shopping and restaurant recommendations, and weather. From the report of Appthority, 50% of the top iOS free apps and 24% of the top iOS paid apps track a user's location. Although users are warned whenever an app intends to capture their locations, they usually choose to allow the permission for the function offered by the app. Apps that over collect location data can be separated into two main types: location service as main function and location service as the auxiliary function. The first type of apps normally ask users for permissions to their location information, while the other app type can collect users' location information without noticing users. The first and the most direct risk is a physical security concern. Users' tracks are easily exposed to those who have users' real-time and accurate location data. Users' habits and customs are easy to be inferred by using simple data mining methods.

Furthermore, solving the data over collection problem is also a research issue in mobile apps. PiOS, presented by M. Egele et al., to detect privacy leaks in iOS applications, used static analysis to detect sensitive data flow to achieve the aim of detecting privacy leaks in ap- plications in iOS. Sharing a similar goal with PiOS, TaintDroid, is a system wide dynamic taint tracking multiple sources of sensitive data. The main strategy of TaintDroid is real-time analysis by leveraging Android's virtualized execution environment. Another secure model via automated validation uses commodity cloud infrastructure to emulate smartphones to dynamically track information flows and actions. This model automatically detects malicious behaviors and sensitive data misuse via further analysis of dependency graphs based on the tracked information flows and actions.

These approaches or techniques mentioned above only focus on monitoring and detecting apps. The prerequisites are that apps already gain permissions from users. However, these solutions only provide methods of monitoring and detecting behaviors of data over-collections. This approach leaves remedying operations to users, such as disabling the permissions of apps or uninstalling those apps. Users have to manually disable permissions of these apps that over collect users' data or uninstall them. Furthermore, running these approaches or tools adds the consumption of energy, which is particularly valuable for smart- phones with limited resources. As a result, the active method of avoid- ing data over collection behaviors in mobile apps is a crucial challenge that needs to be solved.

1.3.3 Concept of Mobile Apps

Mobile apps were originally developed to offer general productivity and information retrieval, including email, calendar, contacts, and weather information. However, with the rapid increment of public requirement, mobile apps expand into lots of other categories, such as games, music, finance, and news.

A lot of people distinguish apps from applications in a perspective of device forms. They think that applications are used on a desktop or laptop, while apps are used on a phone or tablet. Nevertheless, this simplistic view is too narrow and no longer the consensus, because apps can be used on desktops, and, conversely, applications can run on phones. At Gartner Portals, Content and Collaboration Summit 2013, many experts and developers participated a roundtable discussion titled "Why an App is not an Application". They proposed that the difference between app and application is not about the delivery mechanism and landed on a consensus that:

App = software designed for a single purpose and performs a single function.

Application = software designed to perform a variety of functions.

From the view of users, they do not care whether it is an app or an application by definition, and they just want to accomplish their tasks easily. Meanwhile, from the view of developers, the question they should answer is not whether they should be building an app or an application, but how they can combine the best of both into something users love.

1.3.4 Brief Introduction of Android and Its Framework

1.3.4.1 A Brief History of Android

Android was founded in Palo Alto, California, in October 2003 by Andy Rubin, Rich Miner, Nick Sears, and Chris White in an effort to develop a smarter mobile device that is more aware of its owner's location and preferences. Then to Google acquired Android Inc. and key employees, including Rubin, Miner, and White, on August 17, 2005. At Google, the team, led by Rubin, developed a mobile device platform powered by the Linux kernel. Google had lined up a series of hardware components and software partners and signaled to carriers that it was open to various degrees of cooperation on their part. On November 5, 2007, the Open Handset Alliance unveiled itself with a goal to develop open standards for mobile devices. This alliance includes technology companies, like Google, device manufacturers such as HTC, wireless carriers such as T-Mobile, and chipset makers such as Qualcomm. Then, on October 22, 2008, the first commercially available smartphone running Android came out with a fantasy name: HTC Dream. Since 2008, Android has seen numerous updates that have incrementally improved the operating system, adding new features and fixing bugs in previous releases. There are some milestones of Android SDK, such as Android SDK 2.0 (Eclair) in 2009, Android SDK 3.0 (Honeycomb) for tablets only in 2011, Android SDK 4.0 (Ice Cream Sandwich) in 2011, Android 4.1 to 4.3 (Jelly Bean) in 2012, Android SDK 4.4 (KitKat) in 2013, and Android SDK 5.0 (Lollipop) in 2014.

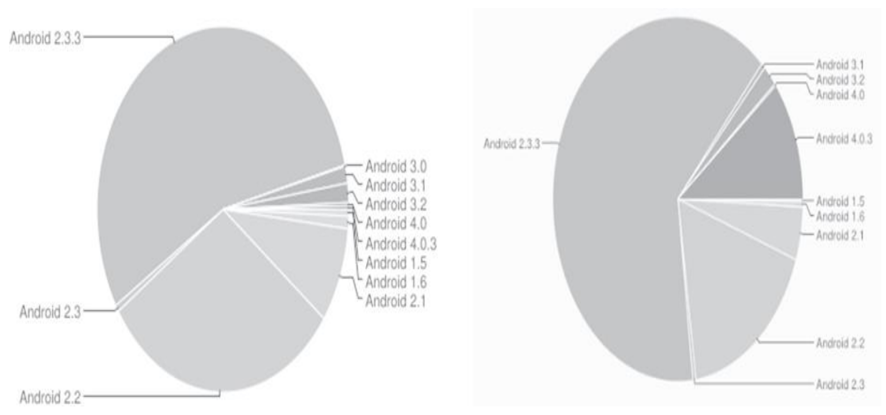

Figure 1.6 Android device distribution in January and July 2012.

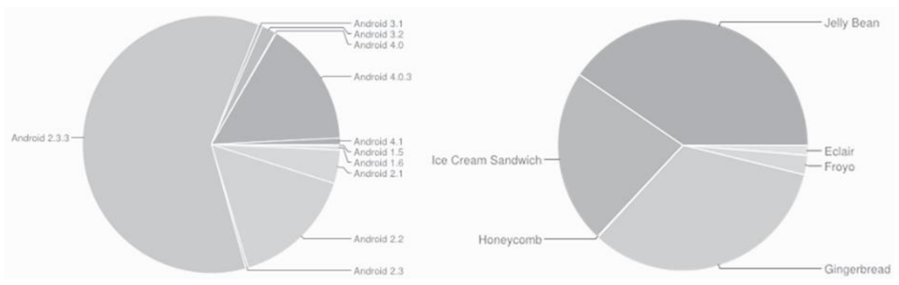

Figure 1.7 Android device distribution in August 2012 and August 2013.

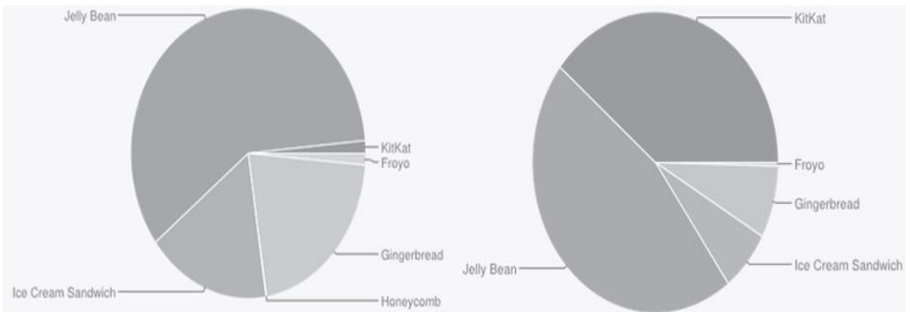

Figure 1.8 Android device distribution in January 2014 and January 2015

1.3.4.2 Android Device Distribution

Fig. 1.6 shows the Android device distributions in 2012. We can see that Android 2.3.3 and 2.2 dominate more than half of the market. Nonetheless, in the second half of 2012, Android 4.0.3 became more and more popular. In August 2013, Android 4.0 and 4.1, named Ice Cream Sandwich and Jelly Bean, respectively, surpassed Android 2.0s and dominated the Android market, as shown in Fig. 1.7. In January 2014, Android 4.1 to 4.3 still dominated the Android market. However, after one year, Android 4.4, named KitKat, rapidly occupied 39.1% of the whole market, as shown in Fig. 1.8.

1.3.4.3 Android SDK

Android SDK is open-source and widely used, which makes it the best choice for teaching and learning mobile development. Android is a soft- ware stack for mobile devices, and it includes a mobile operating sys- tem, middleware, and some key applications. As shown in Fig. 1.9, there are Linux kernel, libraries, application framework, and applications and widgets, from bottom to top. We will introduce them one by one.

The Linux kernel is used to provide some core system services, such as security, memory management, process management, power management, and hardware drivers. These services cannot be called by Android programs directly and is transparent to users. The next layer above the kernel is the native libraries, which are all written in C or C++. These libraries are compiled for the particular hardware architecture used by the mobile devices. They are responsible for handling structured data storage, graphics, audio, video, and network, which only can be called by higher-level programs. Meanwhile, Android run- time is also on top of the kernel, and it includes the Dalvik virtual machine and the core Java libraries.

What is Dalvik? Dalvik is the process virtual machine in Google's Android operating system, which specifically executes applications written for Android. Programs are written in Java and compiled to bytecode for the Java virtual machine, which is then translated to Dalvik bytecode and stored in .dex and .odex files. The compact Dalvik executable format is designed for systems with limited resources.

The application framework layer provides the high-level building blocks used for creating application. It comes preinstalled with Android, but can be extended with its own components as needed. We will introduce some basic and important building blocks of Android.

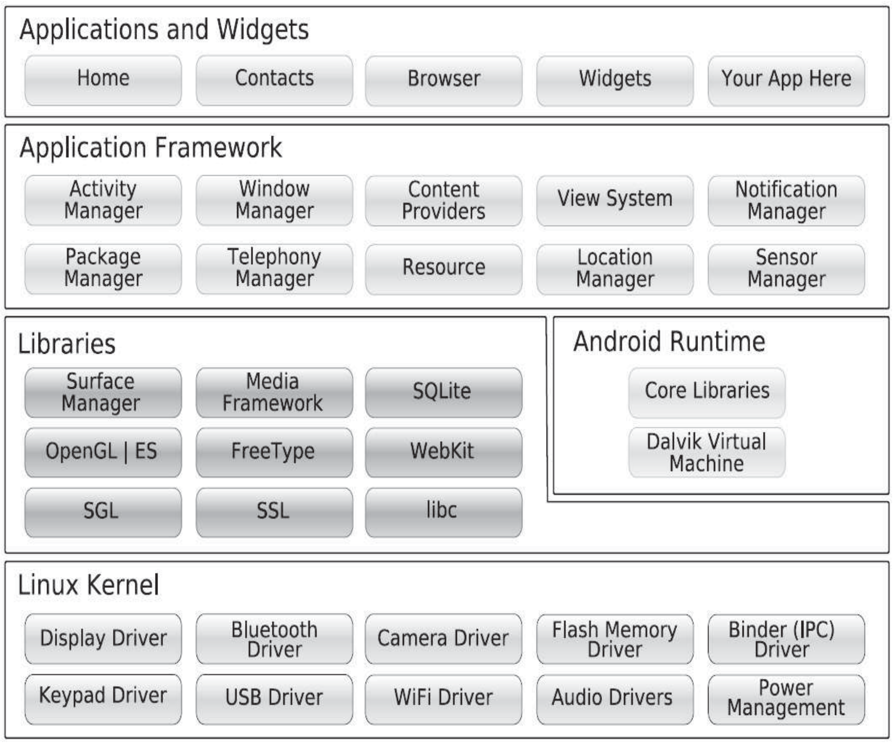

Figure 1.9 Android system architecture.

Activity. An activity is a user interface screen. A single activity defines a single screen with a user interface, and it defines simple life cycle methods like **onCreat, onResume, and onPause** for handling interruptions. Furthermore, applications can define one or more activities to handle different phases of the program.

Intent. An intent is a mechanism for describing a specific action, such as “pick a photo”, or “phone home”. In Android, everything goes through intents, and developer, have plenty of opportunities to replace or reuse components. Intents can be implicit or explicit. An explicit intent can be to invoke another screen when a button is pressed on the *Activity* in context. An implicit intent is when you create an intent and hand it off to the system to handle it.

Service. A service is a task that runs in the background without the user’s direct interaction. In fact, it does the majority of processing for an application. Developers can sub-class the *Service* class to write their own custom service.

Content Provider. A *Content* provider is a set of data wrapped up in a custom Application Programming Interface (API) to read and write it. This is the best way to share global data between applications. The content provider provides a uniform singular interface to the content and data and provides a consistent inter- face to retrieve/store data via RESTful model supporting create, read, update, and delete (CRUD) operations.

An *Android Emulator*, as shown in Fig. 1.10, called Android Vir- tual Device (AVD), is essential to testing Android app but is not a substitute for a real device. AVDs have configurable resolutions, RAM, SD cards, skins, and other hardware. If you have installed Android SDKs, the AVD Manager can allow you to create AVDs that target any Android API level.

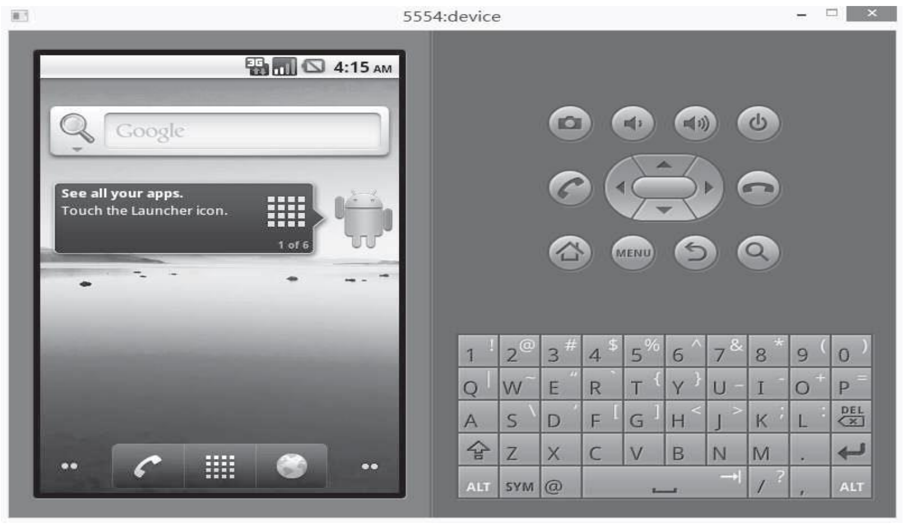

Figure 1.10 Android Emulator.

An Android emulator has the following basic functions:

- Host computer’s keyboard works as keyboard of device.
- Host’s mouse acts as finger.
- Connecting to the Internet using host’s Internet connection.
- Buttons: Home, Menu, Back, Search, Volume up and down.
- Ctrl-F11 toggle landscape to portrait.
- Alt-Enter toggle full-screen mode.

However, emulators have some limitations. They do not support for:

- Placing or receiving actual phone calls. USB and Bluetooth connections.
- Camera or video capture as input. Device-attached headphones.

- Determining connected state.
- Determining battery charge level and AC charging state.
- Determining SD card insert or eject. SD card is a nonvolatile memory card used extensively in portable devices.
- Simulating the accelerometer.

Then we will introduce the process of producing an Android app. In Fig. 1.11, an android app is written in Java and generates .java file. Then *javac* compiler .java reads source files and transforms java code into byte code. Then Dalvik takes responsibility for handling these byte codes combining with other byte codes for other .class files, and generates classes.dex. At last, classes.dex, resources, and AndroidMan- ifest.xml cooperate and generate an .apk file, which is a runnable Android app.

Every Android app must have an AndroidManifest.xml file in its root directory. The manifest presents essential information about the application to the Android system, information the system must have before it can run any of the application's code. The AndroidMani- fest.xml file names the Java package for the application and describes the components of the application, including the activities, services, broadcast receivers, and content providers that the application is composed of. The file also names the classes that implement each of the components and publishes their capabilities. These declarations let the Android system know what the components are and under what conditions they can be launched.

Furthermore, AndroidManifest.xml file determines, which processes will host application components, and it declares which permissions the application must have in order to access protected parts of the API and interact with other application. The file also declares the permission that others are required to have in order to interact with the application's components and lists the instrumentation classes that provide profiling and other information as the application is running. These declarations are present in the manifest only while the application is published. It declares the minimum level of the Android API that the application requires, and it lists the libraries that the application must be linked to.

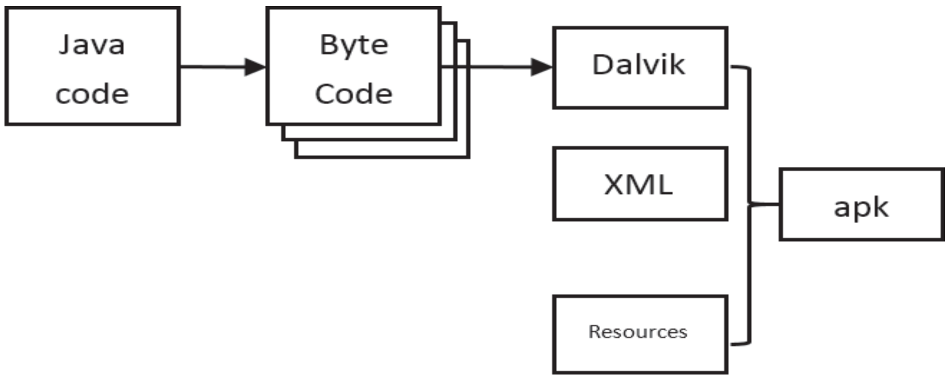

Figure 1.11 Process of producing an Android app.