Jupiter: Transparent Augmentation of Smartphone Capabilities through Cloud Computing

Yao Guo, Lin Zhang, Junjun Kong, Jian Sun, Tao Feng, Xiangqun Chen
Key Laboratory of High-Confidence Software Technologies (Ministry of Education),
School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China
{yaoguo, zhanglin08, sunjian08, kongjj07, fengtao09, cherry}@sei.pku.edu.cn

ABSTRACT

Smartphones have become more and more popular, however the storage and computation limitation remains a big obstacle. With cloud computing and data centers capable of providing almost unlimited storage space and computing capability, we believe that it is possible to augment the smartphone capabilities with the support of cloud computing. This paper proposes Jupiter, a framework to augment the smartphone capabilities with the support of cloud computing. Jupiter can provide transparent user experiences to mobile users in application management and data storage. With the help of virtual machine technology, Jupiter is also able to launch desktop applications on smartphones when necessary. We have implemented a prototype of Jupiter on Android phones and demonstrated its feasibility through a couple of case studies.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]: [Client/server]

General Terms

Design, Experimentation

Keywords

Smartphone, Android, application store, virtual machine

1. INTRODUCTION

Smartphones have become more and more popular in the past couple of years. Although some hardware configurations of many smartphones have been comparable to a ten-year-old desktop PC, the storage and computation limitation is still a great challenge for smartphones if it aims to become a substitute platform for traditional personal computers. Although many smartphones boasts processors up to 1GHz, their memory and storage remains limited compared to laptops, desktops and servers.

On the other hand, cloud computing and data centers are able to provide almost unlimited storage and computing abilities. Although smartphones are able to access cloud storage through third-party providers and also application libraries such as Apple's AppStore and Google's Android Market, these cloud computing abilities can only be accessed through web browsers or special applications.

This paper aims to employ cloud computing as a transparent extension of smartphones, such that cloud storage and computing resources can be accessed transparently from the smartphones. We focus mainly on transparent application management mechanisms on smartphones, while we also develop a transparent file system for data storage as the underlying infrastructure.

1.1 Challenges for Application Management

Smartphone application markets are provided by different organizations, including Apple's AppStore for iPhone, Android Market of Google, and Mobile Market from China Mobile. On these application markets, users are facing at least thousands of applications available for them to choose from.

Take Android Market as an example. On one hand, Android users are excieted that there are so many applications available. On the other hand, they are easily overwhelmed by the flood of applications, which makes software management a heavy burden. Although earlier users of smartphones such as iPhone and Android are normally technology-savvy, as more and more smartphones are distributed in the market, average users are not experts with the technology any more, thus many challenges lies ahead them:

- **Application Installation** - It is time-consuming

MobiHeld '11, October 23, 2011, Cascais, Portugal.

for average smartphone users to locate the suitable application for a given function or feature. For example, there are various video formats: in order to play a specific video, a user might have to try out several players before choosing the correct one.

- **Application Uninstallation** - Because most smartphones have limited storage, users have to remove some infrequently-used applications manually in order to release necessary storage space for the installation of a new application. However, they might want to restore these applications later.
- **Application Upgrading** - When new versions of an installed application is available, users are sometimes notified explicitly by the Android Market or the application itself. Many applications do not push new version notifications to users. Even the notification comes, some expertise is required in the upgrading process, and the upgrading frequencies are also relatively high, especially for Android applications.

On the other hand, although applications are more than abundant on Android smartphones, because of resource limitations, there are still many desktop applications that are necessary for users, but difficult to be re-developed on or ported to smartphones, for example, Microsoft Office compatible applications for documents processing, Adobe Acrobat for PDF editing, and some professional applications for video/audio processing.

Furthermore, applications are not signed or certified in the Android Market, thus applications can be released without any approval, which could result in great damages as more and more viruses/trojans emerging and many critical applications (such as banking and shopping) running on smartphones.

1.2 Jupiter Overview

To address the above problems, this paper presents Jupiter, a mechanism to augment the capabilities of smartphones transparently with the support of cloud computing. One or more cloud servers can be employed as the extension of the storage and computing capabilities of smartphones. The cloud provides storage of applications and user data, and also provide virtual machines to execute large applications that cannot be accommodated on smartphones due to resource constraints.

Jupiter aims to achieve the following goals.

1.2.1 Transparent mobile application management

We develop a customized application library (AppLib) as the storage of pre-certified applications. Applications (.apk) are processed before they are added to AppLib, such that the whole meta information can be extracted and converted to a searchable format. Human involvement is sometimes needed for this process.

Application installation and upgrading are supported via AppLib. Applications demanded by users are searched and compared, only the most appropriate one will be retrieved, installed and launched. The whole process progresses automatically without users' participation.

To make application upgrading transparent to the user, new versions of applications will be updated in AppLib (either automatically or with human interference). On the smartphone, new versions will only be loaded when the applications is executed next time, and the whole upgrading process will be invisible to users.

When the phone storage is full, infrequently-used applications will be removed without the notice of users in order to release storage space for further use. Configurations and data of the uninstalled application will be kept in the transparent date storage mentioned below, and will be restored automatically when the same application is requested next time (the re-installation will also be transparent to the user).

1.2.2 Transparent data storage

We developed a mobile file system in the user space named TransFS, which provides transparent data storage for Android applications. Both application configurations and data can be stored at the server-side and accessed transparently through TransFS.

TransFS enables applications to access remote data with exactly the same interfaces of using locally stored data. It takes advantage of the vast storage capability of the cloud to provide almost infinite storage for smartphones.

1.2.3 Transparent desktop application execution

For large applications that cannot be executed on smartphones, or desktop applications that do not have mobile counterparts, we employ the virtual machine technology on the server, with virtual network computing (VNC) capabilities implemented on the Android, we can allow users to execute desktop applications just as they are executed on the smartphones. The applications are pre-stored in separate storages and indexed through the Jupiter manager.

We have developed a prototype system for Jupiter, which runs on HTC Magic smartphones with Android 1.6 platform. The prototype also includes the TransFS prototype which is implemented on Android as a user-space file system with FUSE. We will demonstrate through two case studies that Jupiter has the potential to provide transparent smartphone augmentation for both storage and computing abilities.

2. JUPITER ARCHITECTURE

The architecture of Jupiter is shown in Figure 2, which contains an Android client and several servers

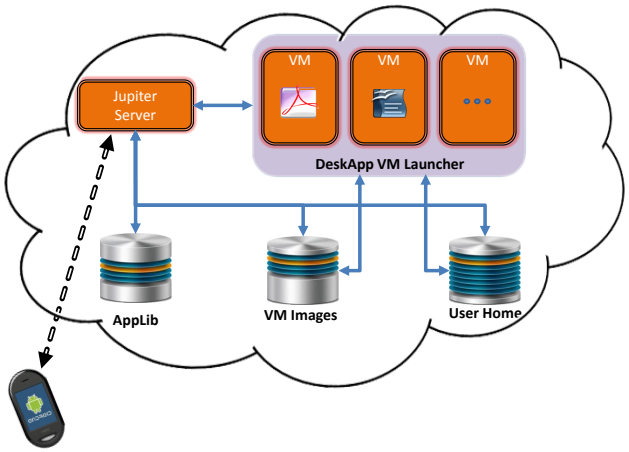

Figure 1: Architecture Overview

in the cloud collaborating with each other.

2.1 Servers in the Cloud

Jupiter employs the servers in the cloud to provide the following functionalities:

- **Storage Servers** - Android applications (AppLib), virtual machine (VM) images and user data are stored on the corresponding servers, respectively. Basic information about users, applications, virtual machines and users' home directories are kept in a MySQL database on the Jupiter Server, together with relationship information among them.
- **Computing Servers** - The Jupiter Server is the key portal of the whole architecture. Requests for applications from Android clients are received and parsed here. Queries are then issued to the application-list database. If a normal Android application is required, the corresponding application (.apk) is retrieved and transferred to the requester.

  If a desktop application is requested, the DeskApp VM Launcher will be signaled and a new virtual machine containing the corresponding application will be created and started.

2.2 The Android Client

There are four layers in the Android platform: Application Layer, Framework Layer, Libraries Layer and Linux Kernel Layer from top to bottom, respectively. Several modifications have been made by Jupiter to the original Android system at each layer, as shown in Figure 2.2.

In order to provide users with a uniform view of normal Android applications(*i.e. Browser*, *Contacts*, *Email*, *Calendar etc.*) and desktop applications(*i.e. Adobe Acrobat* and *Open Office*), the Launcher at the Application Layer is modified, as well as the Package Manager at the Framework Layer. The Jupiter Client is developed to communicate with the Jupiter Server in the cloud. A modified VNC client is also provided for users to connect with running virtual machines on the DeskApp VM Launcher.

There are two ways to load an application: (1) By choosing an application in the Android Launcher: Jupiter will maintain a recommended list of applications, which are stored in AppLib and will be loaded through a click (downloading and installing, if necessary, will be invisible to the user). (2) "On-demand" loading through another application. For example, when the user tries to view/edit an attachment file in an email client, Jupiter will load the corresponding viewer/editor automatically from the AppLib based on the file info .

Desktop application can also be loaded similarly through the above two methods.

In the layers of Linux Kernel and Libraries, we developed a file system named TransFS in user-space by utilizing the FUSE module in the kernel space. With the help of TransFS, users can access their data stored in remote servers as if the data is located in the local smartphone. In Jupiter, there is a home directory for each user on the remote server, which serves as the unlimited storage space extension. Files that are frequently and recently used are cached locally on the phone. This mechanism also benefits the DeskApp VM Launcher, because user data, which is significant and necessary for applications running in virtual machines on the DeskApp VM Launcher, can be easily mounted to corresponding virtual machines for further accesses and manipulations.

3. PROTOTYPE IMPLEMENTATION

The Jupiter client is implemented on an HTC Magic phone running Android 1.6, with MSM7200A 528MHz ARM11 processor, 288MB RAM and 512MB ROM.

We use desktop PCs as our server in the prototype implementation, of course it could be easily ported to high-performance servers to enhance its storage and computing abilities.

3.1 Message Passing in Jupiter

Jupiter is message-driven: many messages will be exchanged between the Jupiter client and the servers in the cloud. Instead of building a new communication protocol based on Socket, we choose to use XMPP[1] , which is an open-standard communication protocol.

The Jupiter server is developed based on a Java

[1]http://xmpp.org/

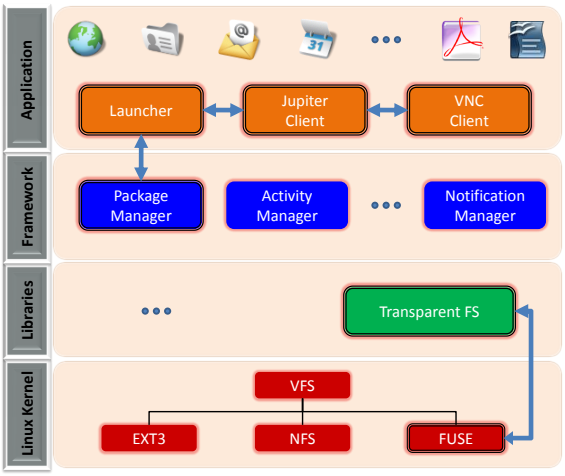

Figure 2: Android Client

library of XMPP named Smack[2] and deployed across servers in the cloud; The Jupiter client is developed based on aSmack[3], which is also a java library of XMPP, available on Android.

The format of messages flowing in Jupiter is JSON[4] (JavaScript Object Notation). It is a lightweight data-exchange format, which is easy for human to read and write, and at the same time for machines to parse and generate. Compared with XML, JSON requires less bandwidth and parsing overhead.

3.2 VNC Implementation

In the VNC implementation, we choose KVM as the virtual machine because it is already built in the Linux Kernel and can be upgraded together with kernels. Furthermore, the size of KVM module is much smaller than other popular virtual machines, such as VMware, VirtualBox and Xen. Finally, and most importantly, it supports incremental storage of virtual machine images.

3.3 TransFS Implementation

In Jupiter, desktop applications are executed remotely on the DeskApp VM Launcher, which is on a server in the cloud. Instead of uploading user data to a specific server in the cloud when necessary, Jupiter provides a new storage model, with which users can conveniently use remote data as if it is local. All user data can be stored remotely, with frequently- and recently-used data cached in the local storage. User data stored in the remote server can be mounted to virtual machines running on the DeskApp VM Launcher easily without any data transmissions.

CyanogenMod[5] is employed in order to build a custom kernel image with the FUSE [6] module enabled, and a new file system in user-space named TransFS is developed on the basis of FUSE at the Library Layer on Android.

TransFS provides the same system call interfaces(*i.e. open(), read(), write(), close(),etc.*) as that traditional file systems. Therefore, it is transparent to applications since they do not need to be modified in order to work correctly. In summary, TransFS provides key support for application management and data/configurationg storage in the Jupiter architecture.

To improve performance, we provide a caching mechanism in TransFS such that frequently used applications and corresponding data are always cached locally, such that user experience can be guaranteed during network disruption.

Detailed implementation of TransFS is not included in this paper due to space limitations.

4. CASE STUDIES

In this section, we provide two case studies with Jupiter to demonstrate its capabilities and feasibility.

4.1 Scenario #1

A new email is received by the Email client with a .pdf file as its attachment. However, applications that can view PDF have not been installed on the phone. As a result, when the user clicks the "**open**" button in the Email client, following steps will be executed:

1. **Lookup** - A request for applications that can view files with specific extension (.pdf) is issued from the Jupiter client on Android to the Jupiter server in the cloud. After the request is received and parsed on the server, a query to the background database is triggered. There might be several applications satisfying the query condition, for example, RepliGo Reader(640 KBytes), ezPDF Reader(3.16 MBytes) and Adobe Reader(1.75 MBytes).

2. **Compare and Select** - According to the current policy in our system, we choose the smallest application, that is, RepliGo Reader. For further extensions, policies can be changed with users' configurations. For example, One can select the most popular one as the optimal solution.

3. **Download, Install and Launch** - Android utilizes a permission-based security model to control accesses to the resources in the system. Applications must require permissions from the user to access certain resources in a standard format

[2]http://www.igniterealtime.org/projects/smack/
[3]http://metajack.im/2010/02/10/xmpp-library-for-android-developers/
[4]http://www.json.org/
[5]http://www.cyanogenmod.com/
[6]http://fuse.sourceforge.net/

Figure 3: Screenshots for Scenario #1

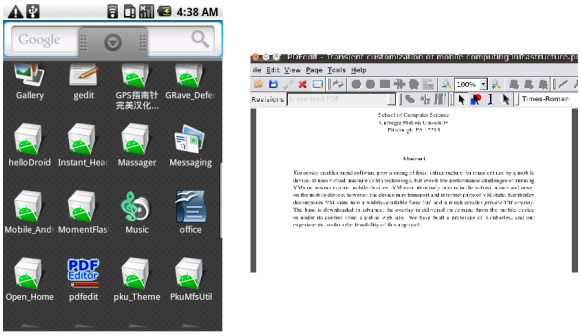

Figure 4: Screenshots for Scenario #2

which is parsed during installation. The Android OS is responsible to allow or deny an access to a specific resource at runtime. In Jupiter, permissions are granted by default in order to install a certain Android application successfully without interference from the user.

Running screenshots for this scenario are shown in Figure 3.

4.2 Scenario #2

Several Emails with PDF attachments have been received from your co-workers. A final document need to be generated from those PDF files and submitted to the manager immediately. However, there are no computers available around. With Jupiter, the user can start a desktop application named PDF Editor in the cloud, to manipulate his documents with his Android phone, as shown in Figure 4.

5. RELATED WORK

Our original ideas are motivated by application on-demand loading, which is first introduced by Gerd Kortuem[10] in 1997. A system named *ACHILLES* is developed for the purpose of software on-demand delivery from stationary servers to mobile computers over wireless network links. With the rapid development and wide spread of smartphones, we believe that it is much more valuable and considerate to apply this mechanism to smartphone platforms.

Several researches have been proposed for running desktop applications on Android. Jay Freeman[7] proposed install and run Debian together with Android on G1 relying on a security hole in the RC29 firmware version. Ghostwalker[8] created instructions for installing Debian on G1 using a custom firmware version developed by JesusFreke[9]. Another effort to install Debian on the Droid was made by Fresh-Meat [10] atop a customized version of the ext2.ko module. By mounting the full Debian Linux distribution into the file system, DebianRunner [8] provides a tool to simplify the installation and execution of Linux desktop applications within Android. However, DebianRunner suffered a lot of performance degradation, and only a calculator can be executed successfully.

It is a straightforward solution to employ cloud computing which provides great storage space and enormous computing capability. CloneCloud[3, 2] creates a cloning of the smart-phone in the cloud. The phone state is periodically or on-demand synchronized; parts of, or even whole of applications are executed in the clone remotely in order to improve performance; and results are returned and integrated back into the phone state. Similar researches are proposed for different purposes, such as energy, performance and security [6, 13, 4, 17, 18, 5, 1, 7, 12, 11, 9, 16].

Rather than relying on a distant cloud, Satyanarayanan *et al.* [14, 15] proposed a new Internet infrastructure which is called *cloudlet-based resource-rich mobile computing*. Mobile devices function as thin clients, with all expensive and significant computation making in nearby resource-rich cloudlets. Cloudlets are decentralized and widely-dispersed Internet infrastructure whose computing and storage capabilities can be leveraged by nearby mobile devices. A mechanism of cloudlet discovery and selection is also provided. As depicted by the authors, it is straightforward to integrate cloudlet with Wi-Fi access point into an easily-deployable entity. However it will be really expensive for cloudlet deployment and management.

6. CONCLUDING REMARKS

This paper proposes Jupiter, a framework to provide transparent augmentation of smartphone capabilities with the support of cloud computing. Jupiter can provide transparent user experiences to smartphone users in application management and data storage. With the help of virtual machine technology, Jupiter is also

[7]http://www.saurik.com/id/10/
[8]http://code.google.com/p/android-roms/
[9]A user from Android-Roms google project
[10]http://alldroid.org/archived/threads/14749.html

able to launch desktop applications on smartphones if necessary. We have implemented a prototype of Jupiter on Android phones and demonstrated its feasibility through a couple of case studies.

The current Jupiter implementation relies heavily on the quality of network connections. Although various network connectivities are supported on Android phones, including GPRS/EDGE for 2G networks, WCDMA for 3G networks and Wi-Fi, we still cannot guarantee persistent Internet connectivity at this stage. In order to mitigate this problem, caching has been considered in Jupiter implementation (e.g., TransFS). When persistent connection does not exist, the users cannot load remote applications or start remote virtual machines, however, users can still use the phone as a standalone device.

Another concern is the network bandwidth limitation. Actual bandwidths in different Internet connections are still limited, even in current 3G networks. Our test with the current prototype shows that if WiFi is available, Jupiter can run smoothly with no interruption. However, for applications larger than 1MB, the loading time via EDGE/3G networks is noticeable to the users.

Jupiter is still in a very early stage, we are currently working on improving many facets of Jupiter including its performance, scalability, user interfaces, etc. We believe that the above limitations will be overcome in the near future with the fast development of new networking technologies, for example, the emerging 4G (LTE) standards.

7. REFERENCES

[1] R. Balan, J. Flinn, M. Satyanarayanan, S. Sinnamohideen, and H.-I. Yang. The case for cyber foraging. In *Proceedings of the 10th workshop on ACM SIGOPS European workshop*, page 87, 2002.

[2] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti. Clonecloud: elastic execution between mobile device and cloud. In *EuroSys '11, Proceedings of the sixth european conference on Computer systems*, pages 301–314, 2011.

[3] B.-G. Chun and P. Maniatis. Augmented Smartphone Applications Through Clone Cloud Execution. In *HotOS*, 2009.

[4] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. Maui: making smartphones last longer with code offload. In *MobiSys '10*, pages 49–62, 2010.

[5] J. Flinn, D. Narayanan, and M. Satyanarayanan. Self-tuned remote execution for pervasive computing. In *HotOS'07*, pages 61–66, 2007.

[6] J. Flinn and M. Satyanarayanan. Energy-aware adaptation for mobile applications. *ACM SIGOPS Operating Systems Review*, 33(5):48–63, Dec. 1999.

[7] A. Fox, S. D. Gribble, E. A. Brewer, and E. Amir. Adapting to network and client variability via on-demand dynamic distillation. *ACM SIGOPS Operating Systems Review*, 30(5):160–170, Dec. 1996.

[8] A. Gupta, A. Rodriguez, and K. Preston. DebianRunner : Running Desktop Applications on Android Smartphones. Technical report, University of Illinois, 10 2010.

[9] X. Jiang, X. Wang, and D. Xu. Stealthy malware detection and monitoring through vmm-based out-of-the-box semantic view reconstruction. *ACM Trans. Inf. Syst. Secur.*, 13:12:1–12:28, March 2010.

[10] G. Kortuem, S. Fickas, Z. Segall, and D. Hall. On-Demand Delivery of Software in Mobile Environments. In *Nomadic Computing Workshop, ICPP*, 1997.

[11] J. Oberheide, K. Veeraraghavan, E. Cooke, J. Flinn, and F. Jahanian. Virtualized in-cloud security services for mobile devices. In *Proceedings of the First Workshop on Virtualization in Mobile Computing*, MobiVirt '08, pages 31–35, 2008.

[12] G. Portokalidis and H. Bos. Paranoid Android : Versatile Protection For Smartphones. *Network Security*, 2008.

[13] A. Rudenko and P. Reiher. Saving portable computer battery power through remote process execution. *Mobile Computing and Communications Review*, 2, 1998.

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

[15] A. Wolbach, J. Harkes, S. Chellappa, and M. Satyanarayanan. Transient customization of mobile computing infrastructure. In *Proceedings of the First Workshop on Virtualization in Mobile Computing*, MobiVirt '08, pages 37–41, 2008.

[16] X. Zhang, J. Schiffman, S. Gibbs, A. Kunjithapatham, and S. Jeong. Securing elastic applications on mobile devices for cloud computing. In *Proceedings of the 2009 ACM workshop on Cloud computing security - CCSW '09*, page 127, 2009.

[17] B. Zhao, B. C. Tak, and G. Cao. Reducing the delay and power consumption of web browsing on smartphones in 3G networks. In *ICDCS*, 2011.

[18] B. Zhao, Z. Xu, C. Chi, S. Zhu, and G. Cao. Mirroring Smartphones For Good : A Feasibility Study. In *MobiQuitous*, pages 1–12, 2010.