oPsai: Seamless Migration of Remote User Experience to Personal Cloud

Hyun Jong Lee
Under the guidance of Minsung Jang and Prof. Karsten Schwan
School of Computer Science, Georgia Institute of Technology
Email: {firsname}.{lastname}@cc.gatech.edu

***Abstract*—As mobile devices are downwardly standarized in the post-PC era, the distinction between devices decays and people become oblivious to the potential of devices. This project implements a user-cleint Personal Cloud for low-power devices, constrained with an ability to local expansion, allowing the device to employ remote resources to leverage physical, geographical limitation. Futhermore, the project extends the idea of Remote Frame Buffer (RFB) [2], integrates Advanced Linux Sound Architecture (ALSA) [3] with UDP server and implements ZeroMQ for its communication channel with the PCloud host to acquire current avaliable destination and functionality of the remote resources. The amalgamation of these three notions is called "*okay, PCloud show as is* (oPsai)". Latter on the paper, simple projector streaming and jukebox applications are deployed to depict the potentiality of PCloud.**

I. Introduction

Today's smart phones give users two highly desirable functions. One is the ability to adapt abundant options in sensors on the devices (e.g., gyroscope, motion detectors) timelessly. Another is access to mobile resources (e.g. cloud storage, streaming videos) wherever they are at any moment. Despite abundant options in sensors on the mobile devices, end-user experiences on the devices, especially on the Android-based platform, have been constrained by short battery cycle, the **scope** of the data available locally [1], and vendor-specific solutions such as Chromecast. This project focuses on addressing these current obstacles by reflecting potential approaches and by building a notion for borderless use of both nearby and cloud resources. To facilitate implementation, distributed software abstractions used by preceding projects allow access to such remote resources. Moreover, the project provides end users with the amalgamation of networked resources available in the current environment [1] by a personalized execution environment from an application point of view.

There has been numerous research projects concerning bringing PC exprience to mobile devices such as Smart VNC [4] and Soul Pad [5]. However, it has been rarely attempted on the other way around. An advance in mobile devices throughout the past five years has destroyed the boundary between mobile and non-mobile devices. However, as mobile devices with wide screens and better resolution crumble, the benefit of being **mobile**, various applications (Apps), and vendor specific solution advents have encouraged people to preserve mobile ecosystem **as-is** while still longing to enjoy the endless extension of devices with PC-like environment.

Most vendor and/or App specific solutions aim for a particular purposoe (e.g. streaming "their" contents via "their" products), have to be manually configured by users and furthermore impose a limitation on expansion of locally usable resources.

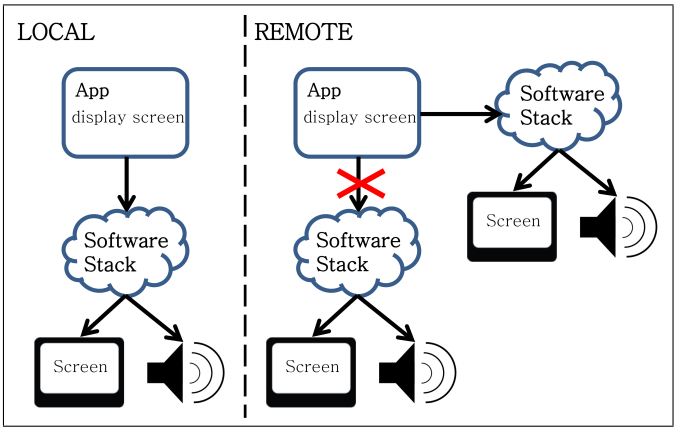

Fig. 1: Traditional point of view from App on its local and remote resources

Existing Apps designed for the local-oriented environment and do not support remote resources. Though a user installs remote solutions, the App is required to recognize certain policies of the resources and therefore demands external application framework at remote end point (shown in Figure 1). Wireless ad hoc system framework at least satisfies the basic functionality but makes existing Apps and/or resources useless. Current remote resource solutions involve redundant system framework, comparable to using devices as an "ad hoc". The approach first presented in 1997 [9] is so obsolete that it does not effectively employ prolific desirable functions of the devices. Moreover, the user has to statically manage the configuration for each resources [7].

In this paper, we present a system library *oPsai*, a runtime daemon porting existing PCloud into Android. The *oPsai* approaches constraints through a transparent, ambient, ergonomic and universally applicable solution. It not only delivers users remote screen, sound, input and countless extension of sensors but also provides current apps the trasparent ambient resource access, by providing PCloud awareness to existing Android environment without a modification. The *oPsai* demonstrate potential approaches that augment a native Android devices innate capabilities, improve performance and

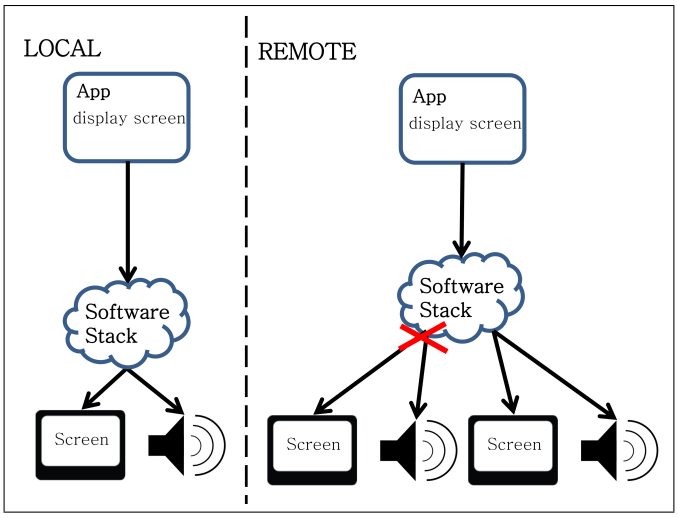

Fig. 2: In oPsai, point of view from App on its local and remote resources

performs borderless interactions with remote resources by integrating two different framework into a standalone system framework shown in Figure 2.

II. Motivating Scenario

This section addresses how oPsai can leverage cloud resources both locally and remotely to deliver end users with countless potential use cases in following scenarios: 1 to 1, N to 1 and 1 to N.

A. Home Usecase

Regardless how much smartphone advances, there are tasks smartphones cannot do but are innate functions as a "phone". Frequent tasks such as checking emails, talking over call and receiving SMS are often conducted on the phone whereas long-term tasks such as file exploring and internet exploring are preferred to be done in the tranditional PC [11]. Remote screen with **1 to 1** scenario effectively fulfills an eager to multitask works for both PC and phone at working environment. The oPsai extends an idea of remote display to remote touchscreen. Without a virtual hardware or system, user

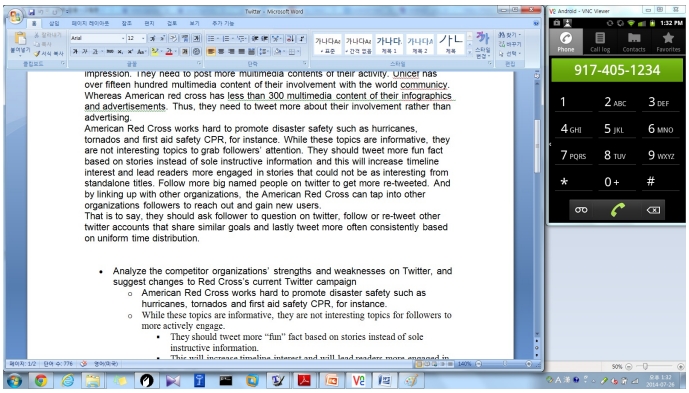

Fig. 3: 1 to 1 remote screen sharing

expriences amalgation of two distinct devices in one integrated familiar environment (e.g. Android-only application on Windows 7) and are disentangled from screen size and resolution of smartphone by seamlessly, dynamically deploying wider cloud screen at need.

B. Public Place Usecase

Backed by PCloud [1], oPsai provides dynamic policy based on SNS. Among all benefits in **N to 1** scenarios, remote menu screen with cloud Point of Sale (PoS) and Jukebox service thoroughly describe practical application in our daily life. Remote menu screen conveys efficient method for static written menu table at public cafe with long standing line.

The following scenario demonstrates small usecase where *N* PCloud hosts communicate to *1* PCloud agent and an user is served with multiple PCloud instances. 1) A user enters a shopping mall where N dinning places are avaliable. 2) PCloud host at each dinning place computes statistical probability of user's tastes based on SNS context and sends the results to PCloud agent in oPsai. 3) PCloud agent pops up a list of suggested menu for each dinning place on the user's smartphone screen. 4) User selects Starbucks cafe for venti green tea frappuccino and oPsai initiates remote screen sharing with PCloud host at Starbucks. 5) Details of the menu along with free deserts for Starbucks Gold card holders are enlisted in user's remote smartphone screen, streamed from Starbuck PCloud host. 6) User picks plain bagel with cream cheese for free desserts, suggested by PCloud host, and automatically proceeds to PoS with amount at the card, verified by oPsai. 7) The oPsai broadcasts the end of transaction, receives expected pickup time, sets pending alarm to remind the user and lastly pushes the order to PCloud host so that remote Starbucks baristas prepare for it.

The scenario, mainly using remote screen sharing, compensates both user and corporation. Dynamic suggestion on the menu of diverse dinning places provides the user innumerable access to nearby places' menu remotely and filters unlikely-chosen choices among the lists. Individually focused service for a user (e.g. free desert for Starbucks Gold card) generates special guest exprience as if multiple hosts are serving only for the user. This is applicable because smartphones, running oPsai, acquire PCloud awareness.

Another desirable application is using a mobile smartphone as a Jukebox. We use smartphone not only for a call but also for music player. Extending the idea of traditional Jukebox, for which one pays a few cents to stream music he likes to listen in certain location for given period of time, we introduce mobile cloud Jukebox service. 1) As user enters Starbucks to pick up the order, oPsai receives lists of enqueued music to stream at the location and is granted to remotely stream music for Gold card holder. 2) Scanning avaliable local and cloud music the user has permission to, oPsai pops up a notification for playing music she likes after current music streaming. 3) She selects "Dear Mr. President" music file from her Dropbox. 4) oPsai enqueues the song to PCloud host and begins to download the file to prepare for streaming the music. 5) After current song ends, she enjoys sweet green tea frappuccino, listening her favorite music.

C. Class Usecase

We own various mobile devices ranging from a 3 inch smartphone to 10.1 inch tablet for different needs. Ironically, we use one device at a time and thereby are wasting possible computational resource of the others. Increasing screen size involves so much redundant devices that we are overwhelmed by the number of devices to carry around during the day. oPsai removes the constraint of screen size using larger screen device as a "dummy" device. Computation is done in original smart phone but "users" can connect to the device to receive screen streaming.

Fig. 4: 1 to N remote screen sharing

1 to N use case is applicable in educational purpose. Most often, in a large classroom, an instructor projects his screen against the front wall. Needless to say, students sitting at back of the room barely read the words on the front screen and consequently, are willing to read on their local devices. With oPsai framework, *N* number of students can connect to *1* number of instructor's device to easily obtain context of the projected screen.

III. Design

Preserving current workflow and environment for existing Apps was the optimal goal of our project. In this section, we discuss about rationales behind design choices of software stack to effectively implement desirable functionality (e.g. remote display, audio and input) yet to hardly modify current framework.

A. Omnipresent Approach

For PCloud aims to consolidate diverse types of devices to introduce an ubiquitous approach to proceed, we choose to adapt universal softwares and protocols at the lowest layer so that current Apps recognize external hardwares as if theirs. Virtual Network Computing (VNC) [2], operated by RFB protocol, is an outstanding example of light cross-platform softwares such that not only those in Linux but also others in Unix variants (e.g. Apple Remote Desktop) still relies on RFB protocol. The notion behind VNC is that every OS manages its own framebuffer and it is easy to duplicate framebuffer across network. In the case of PCloud, mirroring framebuffer dynamically and platform mutuality are keys to encompass variety of devices.

Since the inauguration of Unix by Dennis MacAlistair Ritchie [19], many systems have evolved, derived or adapted similar architecture from the Unix. As a result, following a standard in the architecture became crucial to porting a program into different system. The Advanced Linux Sound Architecture (ALSA) [3] is Linux common audio system that interacts with kernel layer and is not Android specific.

The universal approach of oPsai by PCloud benefits us to implement the software in variety of devices, ranging from Android to Tizen, and consequently defines an ubiquitous conception deriving from common software and applicable to divergent mobile environment.

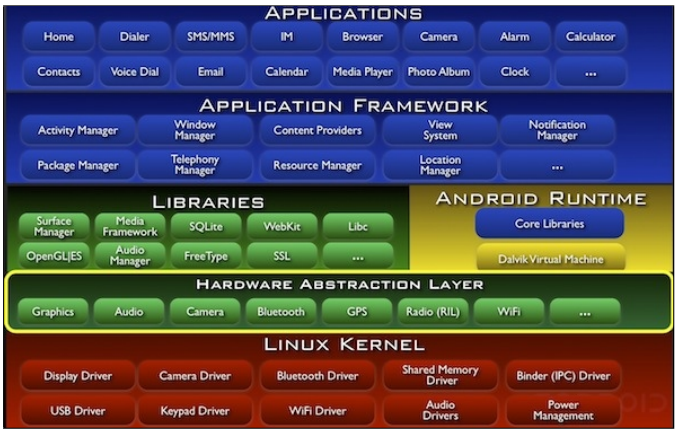

Fig. 5: Android Architecture

B. Layer Constraints

For a demonstration purpose, we implemented the software on Android. Though Android is built on Linux kernel (ver. 3.2) and Unix operating system, it heavily relies on Java running time machine called Dalvik and prevents programs in application layer from directly contacting with libraries and hardware kernels in different layers. For that reason, we choose to substitute a few libraries, which interact with Hardware Abstraction Layer (HAL).

IV. Implementation

This section provides a brief overview of the libraries that are used and supported in this project and rationale behind each choice.

A. System Architecture

In order to minimize an interference between existing Apps and current Android environment, all the libraries are implemented at system library space to communicate in parallel mode with different components.

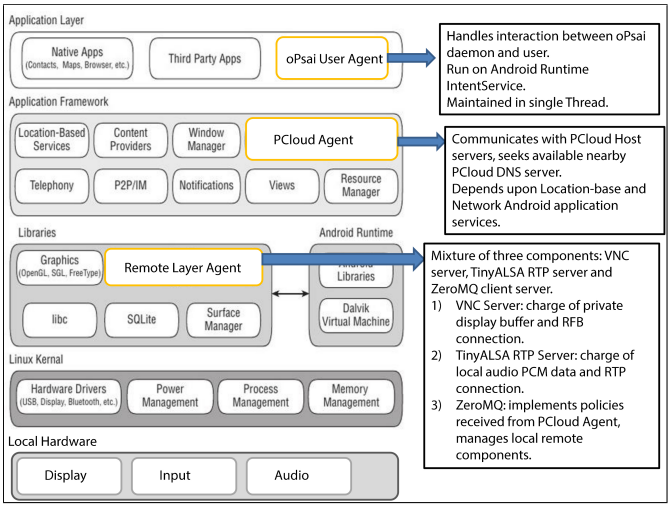

Fig. 6: oPsai general system architecture

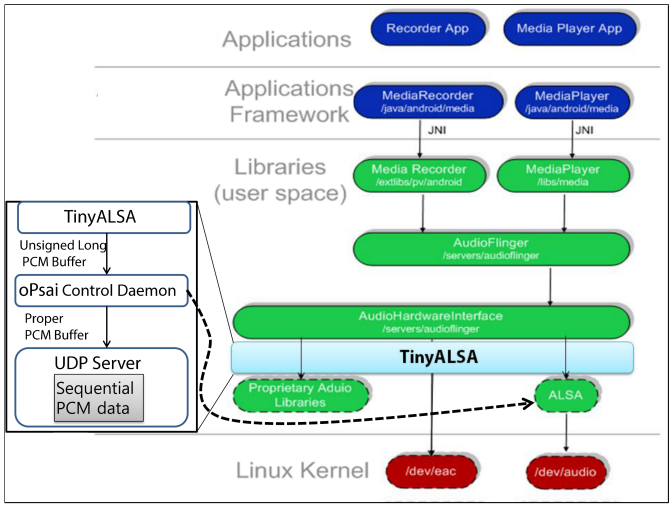

Fig. 7: TinyALSA Subsystem Architecture

1) LibVNCServer: LibVNCServer [6] version 0.99 is a programming framework that enables implementation of VNC server in Android system layer. VNC server runs independently from SurfaceFlinger and is responsible for synchronizing frames with remote frame buffer and inputs from client. Since the server is running on CPU process, it does not have access to buffer in GPU. Therefore, it maintains traditional Linux frame buffer in itself to leverage correct frames of display. For purpose of discussion, we implemented Pure color 24 bit depth with RGB 888 and Pure color 24 bit depth with BGR 565 for each test devices.

2) TinyALSA: TinyALSA is a small library that interacts with ALSA kernel and AudioFlingerPolicy in Android. TinyALSA is from Android Open Source Project and has been implemented throughout different versions. It relies on traditional ioctl system calls to interact with ALSA that is different from using ALSA snd-pcm system call. TinyALSA in oPsai includes audio control daemon which directs a behavior of audio to either remote or local speaker and uses small RTP server to stream raw PCM data to remote ALSA as shown in Figure 6. Theoretical approach to overcome UDP disadvantage is taken to guarantee sequential stream of data. Each PCM data with period size 1024 is accumulated in its local buffer untill at least fifty percent of them are delivered. Cyclic sequential transfer avoids hodgepodge of data, bypasses disadvantage of UDP without running expansive daemon such as PulseAudio yet successfully and fluently outputs audio data to remote ALSA client.

3) ZeroMQ: ZeroMQ version 3.4 is a high performance asynchronous messaging library aimed at scalable distributed and concurrent applications [12]. The library seeks nearby PCloud host using a socket-style API. Different from TinyALSA, the library uses TCP socket to ensure certain reliability. PCloud host, redirected by PCloud DNS, functions as a middle man between oPsai and PCloud resources. The following chart (Figure 7) describes systemetic procedure among three nodes.

	PCloud host	oPsai	PCloud resource (client)
All time on	ZMQ broadcast message	ZMQ receive message & broadcast message if the device is in GaTech longitude and latitude and wake-up (screen turn on).	ZMQ client running for syscall
0		Check condition and broadcast message to the server	
1 (handshake)	Receives message and send echo		
2		Receive echo and sends ack	
3	Receives ack, looks up list of available resource and sends the list		
4		Receives the list and displays the list to user Opsaizmqserver -> opsaiclient (java)	
5		User select the resource, opsaiclient sends signal to opsaizmqserver, zmqserver request to server.	
6 (vnc and sound differs, vnc -> tablet is server & let all others come in, sound -> stream)	Receives the request, sends an invocation to daemon and sends syscall to client with address and port to connect.		
7		Receives address and port, invokes the opsaivncserver	Invoke vncviewer address:5901(default port) –compresslevel 9
8	Waits for end call	Control hands over to the client	Takes control the server
-	Waits for end call	Checks running status of vncserver	Checks running status of vncclient
9	Either daemon or client sends end call, sends to the other	-	-
10		Receive end call and kill the vncserver, ZMQ still alive	Receive end call and kill vncclient, ZMQ client still running

Fig. 8: ZeroMQ based communication scenario

4) Private Framebuffer Multi-Threading: Private Frame Buffer (PFB) is the technique used to avoid collision with SurfaceFlinger in Android system layer. Adreno by Qualcomm and PowerVR by Imagination Technologies are adopted common GPUs in mobile devices. Built on pure Linux operating system, Android previously managed framebuffer in /dev/-graphics/fb0. However, as GPUs' computation power advance, Android team announced Gralloc, sharing memory across process boundaries, synchronizing access to buffers, and pairing the appropriate consumer with the producer [13]. For demonstration purpose, armeabi and x86 architectures have been intentionally used to avert from undesirable confliction with synchronous I/O calls. Relying upon a processor, VNC server has to construct its private frame buffer, which maps particular portion of memory (Ashmem) or saves display frame at the moment (ioctl), to autonomously manage visible frames to both local and remote display. Private Frame Buffer Muti-Threading (PFBM) is another technique utilized in oPsai project to fluently stream simultaneously a large set of pixel change . It uses given number of multithreads for a line of

vertical pixel (ABS-X) per thread to faciliate live streaming such as watching video.

```
#define NUM_CPU 4
#define Y ((4 * 800) / NUM_CPU)
pread(fb_des, raw + (1280 * (Y * thread_id))
        , 1280 * Y, 1280 * (Y * thread_id));
```

B. PERFORMANCE EVALUATION

As shown in the Figure, PFBM on the average takes less time to read and transfer a frame in given time.

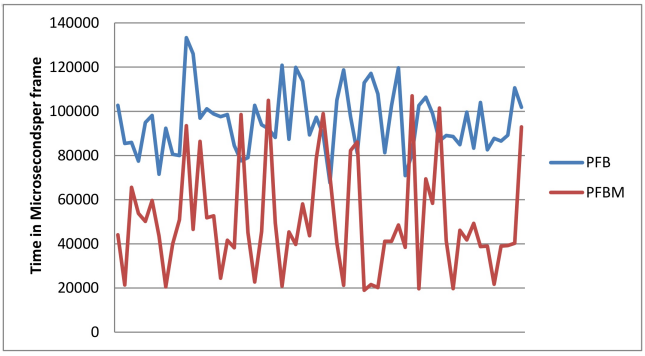

Fig. 9: Performance comparison between PFB and PFBM

V. RELATED WORK

The initial goal of this project was to integrate existing PCloud notion [1] into mobile devices, especially on Android. At some extent, we achieved the goal as getting closer to "seamless" deployment of remote resources, increasing scope of accessible data and confirming feasible solutions to previous problems. Reversing precedent research, such as SmartVNC [4] and Remote Desktop [16], shares common denominator with Android as a server platform [17] and Remote Control of Mobile Devices [18]. The former research focused on centralizing Android server with multiple clients as applied in our 1 to N notion, and the latter platform resembles running VNC server in system layer.

VI. FUTURE WORK

Current setting of PFBM is rather static and requires adjustment by user for different scenario. VNC optimization process thru compression will resolve the issue of dynamic policy enforcement. Though PCloud agent and TinyALSA are implemented, audio stream is not audible state and entails processing optimization. While designing middleware library, we noticed that there might be security issues with these abstract layers such that none of our daemons governed by Android security policy and privacy interference [14] in application layer cause undesirable results.

As noted in the introduction, mobile devices encompass pleantiful number of sensors. As the era of *Internet of Things* approaches, we expect countless doable, yet practical use cases for improving our daily life. Geoscope for instance may gather population migration during the day time anonymously. Temperature sensors from single-board computer extend scope of data for mobile devices through webs of PCloud. Future work will concentrate upon harmonic integration with external extension into single mobile device so that exisiting environment can be brough to new contextual area **As Is**.

VII. CONCLUSION

This project extends and combines existing infrastructures to deploy remote resources at a need without modification of Apps yet dynamically grants an access to the resources based on geological, social state. The demonstration and performance evaluation establish that low-power devices with much of hardware constraints can satisfactorily expand **scope** of hardwares utilizing remote display, audio and input devices over the network and confirms potential of universal approach taken by PCloud.

ACKNOWLEDGMENT

I wish to thank to Professor Karsten Schwan and Minsung Jang for giving priceless advice, intuitive direction and thorough review on designs I made about research. It has been an honor to serve two profound computer science scholars.

APPENDIX

The source code for this project is available at https://github.com/leecom3025/{opsai-demo-client}, {opsai-x86}, {tinyalsa}, {opsai-armeabi}

REFERENCES

[1] Minsung Jang, Karsten Schwan, Ketan Bhardwaj, Ada Gavrilovska, Adhyas Avasthi. *Personal Clouds: Sharing and Integrating Networked Resources to Enhance End User Experiences*. Infocom 2014.

[2] Tristan Richardson, John Levine. *The remote framebuffer protocol*. 2011.

[3] Jaroslav Kysela, ALSA team, *Advanced Linux Sound Architecture*. 1998.

[4] Cheng-Lin Tsao, Sandeep Kakumanu, and Raghupathy Sivakumar. *SmartVNC: an effective remote computing solution for smartphones*. Proceedings of the 17th annual international conference on Mobile computing and networking. ACM, 2011.

[5] Ramn Cceres Casey Carter Chandra Narayanaswami Mandayam Raghunath. *Reincarnating pcs with portable soulpads*. Proceedings of the 3rd international conference on Mobile systems, applications, and services. ACM, 2005.

[6] Johannes E. Schindelin, *LibVNCServer*. GNU. 2003

[7] Meguerdichian, Seapahn, Farinaz Koushanfar, Gang Qu, and Miodrag Potkonjak. *Exposure in wireless ad-hoc sensor networks*. In Proceedings of the 7th annual international conference on Mobile computing and networking, pp. 139-150. ACM, 2001.

[8] M. Jang and K. Schwan, *Stratus: Assembling virtual platforms from device clouds*. in IEEE CLOUD, 2011, pp. 476483.

[9] Hodes, Todd D., Randy H. Katz, Edouard Servan-Schreiber, and Lawrence Rowe. *Composable ad-hoc mobile services for universal interaction*. In Proceedings of the 3rd annual ACM/IEEE international conference on Mobile computing and networking, pp. 1-12. ACM, 1997.

[10] Chun, Byung-Gon, and Petros Maniatis. *Augmented Smartphone Applications Through Clone Cloud Execution*. In HotOS, vol. 9, pp. 8-11. 2009.

[11] Karlson, Amy K., Brian R. Meyers, Andy Jacobs, Paul Johns, and Shaun K. Kane. *Working overtime: Patterns of smartphone and PC usage in the day of an information worker*. In Pervasive computing, pp. 398-405. Springer Berlin Heidelberg, 2009.

[12] Pieter Hintjens. *The ZeroMQ Guide - for C Developers*, ZeroMQ Community, 2010.

[13] Android Open Source Project. *Graphics*. Google Inc. 2009.

[14] Gibler, Clint, Jonathan Crussell, Jeremy Erickson, and Hao Chen. *AndroidLeaks: automatically detecting potential privacy leaks in android applications on a large scale*. Springer Berlin Heidelberg, 2012.

[15] Yi, Won-Jae, Weidi Jia, and Jafar Saniie. *Mobile sensor data collector using Android smartphone*. In Circuits and Systems (MWSCAS), 2012 IEEE 55th International Midwest Symposium on, pp. 956-959. IEEE, 2012.

[16] Jadhav, Archana, Vipul Oswal, Sagar Madane, Harshal Zope, and Vishal Hatmode. *VNC Architecture Based Remote Desktop Access Through Android Mobile Phones*. International Journal of Advanced Research in Computer and Communication Engineering 1, no. 2 (2012).

[17] Toyama, Masashi, Shunsuke Kurumatani, Joon Heo, Kenji Terada, and Eric Y. Chen. *Android as a server platform*. In Consumer Communications and Networking Conference (CCNC), 2011 IEEE, pp. 1181-1185. IEEE, 2011.

[18] Villan, Angel Gonzalez, and Josep Jorba. *Remote control of mobile devices in Android platform*. arXiv preprint arXiv:1310.5850 (2013).

[19] Pioneer Programmer Shaped the Evolution of Computers, Wall Street Journal, October 14, 2011, p.A7.