An Approach to Explore Mobile Software Engineering Advances in Cloud Computing Environment

Weider D. Yu **Hongbin Yuan**

Computer Engineering Department
San Jose State University
San Jose (Silicon Valley), California 95192-0190, USA
EMAIL: Weider.Yu@sjsu.edu

Abstract – Mobile software, helped by the astonishing success from Apple iPhone's debut to Google Android's follow-up, has become a new and hot growth area in the software computing industry. From a software engineering point of view, cloud computing with Service Oriented Architecture (SOA) is a potential paradigm shift following the major paradigm shift from mainframe based computing to client-server computing in the early 1980s. The purpose of the paper is to perform a study on the state of the art of cloud computing technology and to understand the effectiveness of applying this technology in the field of mobile software engineering. To facilitate the research study, we built a music streaming application prototype system based on Android platform and Google App Engine to explore the impact and effect of cloud computing in mobile software environment.

Index Terms – Cloud Computing, App Engine, Mobile Software Engineering, Android Platform, Streaming, Service Oriented Architecture

I. INTRODUCTION

Cloud computing is Internet based computing. From a software engineering point of view, it is a paradigm shift following the shift from mainframe to client-server in the early 1980s. The similarity with client-server architecture is that cloud computing is still sharing resources. The difference is this "sharing" would be provided based on user demand and charged on consumed quota. The advantage is that it reduces the expertise and cost for establishing and maintaining the server infrastructure from users, thus allowing them to focus on business application development. The disadvantage is that it may lock the user to a specific cloud service provider due to the poorer portability caused by the lack of a standard interface, thus brings up feasibility and security concerns to many potential users. [3]

Mobile software, helped by the astonishing success starting from Apple iPhone's debut to Google Android's follow-up, has become a new software growth area. While mobile software carries inherent mobility and accessibility, it is also limited by the physical device size and resource constrains (both storage and computing power). Compared to the desktop PC world, the mobile OS is even diversified and segmented. The major OS with a considerable installation base include iPhone iOS, Android, BlackBerry, Symbian, Windows Mobile, BREW, and Palm OS.

When cloud computing becomes mobile, it means the mobile device tries to access a shared computing power or storage on demand. On one hand, cloud computing is an expansion of the mobile device to supplement its relatively limited computing resources, but on the other hand, mobile cloud computing is also limited by slower or interrupted Internet connections in 3G or Wi-Fi networks. [4]

The solutions are being developed and evolved. One of the promising technologies is HTML5 as it offers off-line data caching and lets cloud-based web applications have the same behavior when the Internet connection goes down. Furthermore, HTML5 enables updating the cached data from server for the changes happened during the off-line period. It aims to smooth the user experience and reduce the server load if the real time information is not critical. Device-based mobile applications can also adopt proactive downloading based on user subscription and device condition and policy settings. To aggregate these features, a "rich-client" or "thick client" agent-based model could be built to provide systematic services to all device-based applications for accessing cloud computing resources. In contrast, "widget" based technique is often used for singleton task which delivering cloud connection and rich user interface. [5][6]

II. ARCHITECTURE, TECHNOLOGIES, AND PROCESS

In this section, we will present the CMPlayer (Cloud Music Player), an Android smart phone music streaming application prototype system with Google App Engine in the backend. The prototype system is served as a major study subject facilitating in exploring the mobile software engineering advances in cloud computing environment.

978-0-7695-4459-5/11 $26.00 © 2011 IEEE
DOI 10.1109/COMPSACW.2011.98

A. Architecture

The CMPlayer adopts the 3-tier client-server architecture and design pattern in which the presentation tier, business logic tier, data storage/access tier are developed and maintained as independent modules so that each modules can be updated or replaced independently. The prototype enables Android smart phone users to access to the Google App Engine (GAE) cloud space for music streaming and uploading services.

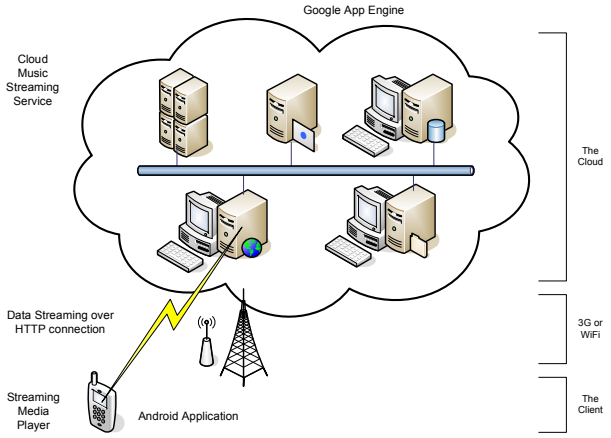

Figure 1: Overall System Architecture

1) Overview

The presentation tier is a standard Android Java application. This is the top level of the application which users can access the system main functionality – "streaming music from cloud", "generating playlist from cloud", and "uploading local music to cloud".

The logic tier is across Internet between two end points: Android smart phone and J2EE web application. It executes the user commands by sending a HTTP request to server end, invoking server end components execution, and retrieving back the result in a HTTP response.

The data tier adopts Google App Engine as the server end framework. It uses GAE Blobstore service for music file storage and uses GAE Datastore service for music metadata storage. Furthermore it uses GAE Jetty Web Server to hold the CMPlayer web application which is the server part of the logic tier.

2) System Building Block

To prototype streaming music from the cloud, CMPlayer has the following building blocks shown in Figure 2.

On the server end, or GAE cloud space, the CMPlayer web app consists of several J2EE servlets for music downloading, uploading, playlist generating, and user authentication services. The web app is hosted in the Jetty HTTP server. Jetty is adopted by GAE framework to serve as a java-based HTTP server and javax.servlet container. GAE provides essential quota-based cloud computing components in which Datastore and Blobstore services are used by CMPlayer web application.

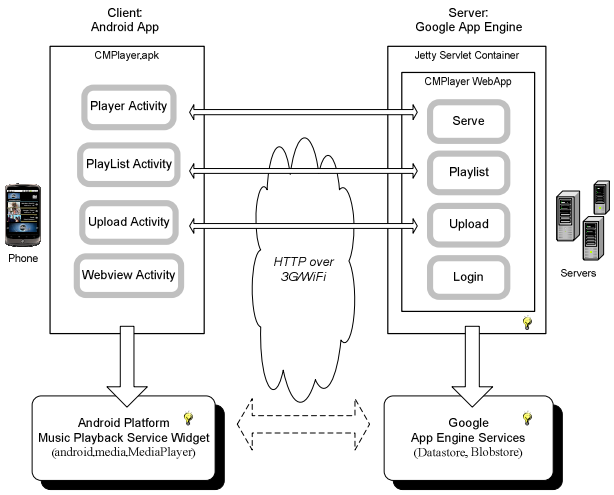

Figure 2: System Building Blocks

On the client side, the CMPlayer client app contains the corresponding activities to server end servlets, and presents a user interface to navigate the end user accessing to the services. The Music Playback Service widget is one of the main OS framework services used by the client application.

Over the cloud, the HTTP protocol is used by both command and data channels. By contrast, the command channel is explicitly established between the CMPlayer client and server components and maintained by the CMPlayer client for command request and information exchange. The data channel, dedicated to music data streaming, is established between the Android Music Playback service and the GAE Blobstore service. The streaming protocol used by the CMPlayer prototype is the HTTP Progressive Download.

3) System Interface

The functioning of modules based on the program flow can be pictorially depicted in Figure 3.

The CMPlayer client is built upon the Android Application Framework and runs as a standalone J2SE Java application on the user's Android smart phone device. Android also supports browser based web applications which can be adopted as the front end to the CMPlayer server. A standalone Java application can accomplish richer user interfaces and flexible application flow control compared to the HTML based web applications.

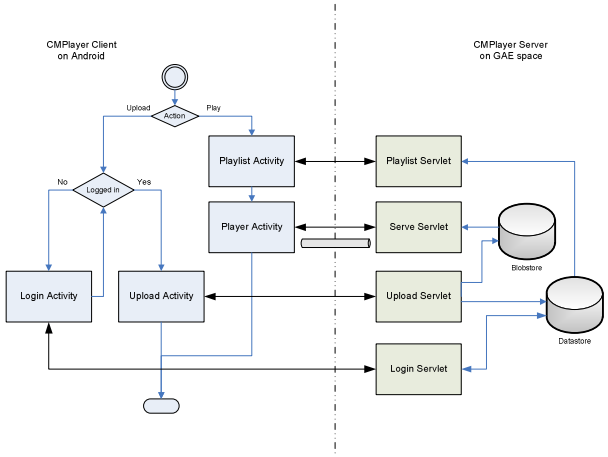

Figure 3: System Interface Diagram

Within the CMPlayer client application, each task is presented to the end user as an activity, which is a basic user interface component in Android Application Framework. User interface rendering is carried by the Android View System and the Window Manager. Activity navigation is managed by the Activity Manger. Music streaming (or playing) is hooked with the Android native Media Player which in turn powered by the underlying Android Media Framework. The music streaming data source, which is pointed to the CMPlayer server, is wrapped in form of Android Content Provider service. The music uploading feature for internet communication and data transfer is powered by an Apache open source project – HTTP Client. Furthermore, user authentication can be directly built upon the Google user account to reach seamless integration between the client and the server as both of them are powered by Google products.

The CMPlayer server is architected as a J2EE web application consisting of corresponding servlets and JSPs to fulfill the client requests. Even though Google App Engine supports both Java and Python server side technologies, using Java API seems to be a nature choice since the CMPlayer client uses Java technologies.

Since GAE can only execute code called from an HTTP request, the CMPlayer client will interact with the CMPlayer server by invoking the server side JSP interfaces. Upon receiving client end HTTP requests, the Jetty web server at the server end will launch corresponding servlets for music upload, download/streaming, playlist population, and user authentication services, and reply back the HTTP response for the command execution results.

Within the CMPlayer web application, servlets act as the logic tier to parse the client requests and make business logic decisions. If necessary, it will communicate with the data tier for the data access services. The Java runtime environment provides APIs for various App Engine services. Among these services, the CMPlayer utilizes the Datastore service for storing and fetching music meta-data, utilizes the Blobstore service for storing and serving the music files, and utilizes the User service for authentication.

B. Technologies

1) Client Technologies

The client development is carried using the Android J2SE technologies and Application Framework. In contrast to iOS, Android is an open mobile platform which allows developers developing application in Java language.

The CMPlayer client application consists of self-defined activities for the user interface design and utilizes the Android system Media Player service for the music playback/streaming. The CMPlayer middle tier networking components are built upon the Apache HTTP Client library which is essentially based on the WebKit native runtime library. The server response data is wrapped in form of XML format so that the Android XML parser is also used in the client application. The rich set of the Android application framework makes it possible to build a rich featured cloud based application in mobile phone device.

2) Server Technologies

The server development is carried using J2EE technologies (JSP/Servlet) and integrated with GAE services. In contrast to Amazon Web Service, the GAE is also hosting our web app directly and providing load balancing, etc.

Similar to Android platform, Eclipse IDE is the preferred development environment for App Engine based web application. The Google Plugin for Eclipse is bundled with the App Engine Software Development Kit (SDK) and facilitates the development and deployment, such as new project wizard, debug configuration, and App Engine deployment.

3) Data-Tier Technologies

The prototype relies on GAE Datastore and Blobstore for metadata and music file storage, respectively. In contrast to Amazon AWS S3, Google's database is not a relational database but with the similar interface, and bears with certain constrains, such as maximum 1MB data object in Datastore and maximum 2GB data object in Blobstore. Even with free quota for experimentation, to enable Blobstore API, developer has to fill in the billing information first – not ideal, but still better than AWS, which has no free quota to kick of the prototype project.

C. Cloud Based Mobile Software Development Process

The Figure 4 shows our cloud based mobile software development lifecycle processes in activity diagram. It covers the various project phases, from the project topic rendering to the prototype origination and development

choice. The procedure shape in green color involved in cloud based activities, either with the Google App Engine Tools or through programming interfaces, which requires upfront research on various cloud based infrastructure, platform, and services.

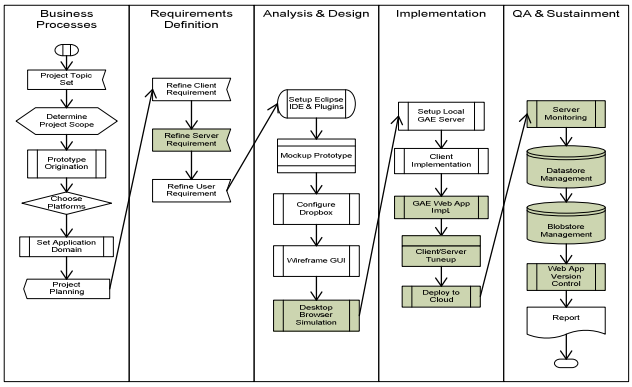

Figure 4: Cloud Based Mobile Software Development Process

III. SYSTEM DESIGN

A. Client Design

The client side design is depicted in the natural navigation order of the tab view for the application tasks: streaming, download, upload, and WebView test bed.

The *PlayerView* activity is to build a music player for music streaming. The main components are VideoView and MediaController which in turn are native widget classes built upon the native MediaPlayer technology.

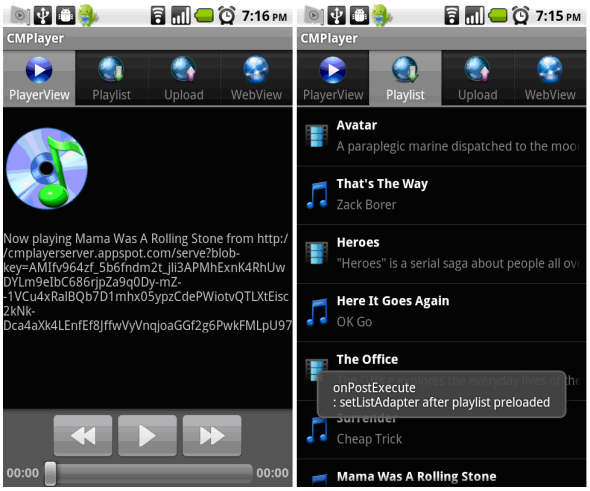

Figure 5: PlayerView and Playlist Activities

The Android MediaPlayer can play remote music file if the server supports the HTTP Progressive Download protocol. To client, this means feeding the player with a streaming URL pointing to the CMPlayer server music file. The streaming URL is composed based on runtime configuration and the blob key inside the song's Record class. Once the song starts play, the streaming will be handled by the media player instance.

The *Playlist* activity is to retrieve list of songs from the CMPlayer server cloud space and presents them in the ListView layout. User can select a song by touching the list item. Upon the clicking event, the active view is switched to PlayerView, and the current song's Record will be set and the play method of the PlayerView activity will be invoked.

This activity is the first view user will see after the application gets launched. We are intended to use this activity to prototype most of the download related tasks. There are some highlights in regarding to building the Playlist activity:

- A list view item adapter is required to attach the song's Record object to the UI list view item;
- A SAX style XML parser is necessary to populate song's Record collection from the playlist XML document;
- Both client and server application are object oriented design but the HTTP transmission requires data marshalling as XML is a common data exchange format for HTTP request/response;
- The networking operations are carried in background with an asynchronous task to prevent interfering with the GUI operations.

Figure 6: Upload and WebView Activities

The *Upload* activity provides a simple user interface allowing user uploading music to the CMPlayer server end GAE Blobstore music database. This activity adopts Android TableLayout for UI representation. The upload task is carried by using the Apache HTTP client Multipart

post method. Once user clicks the submit button, the server end corresponding JSP URLs will be invoked for the Web-Form data transmission. Since it usually takes longer time for upload, the internet operations are invoked in an asynchronous task.

B. Middle-Tier Design

The middle tier includes the CMPlayer client application that has networking capabilities to interact with the client user interface and send commands to the CMPlayer server.

We have seen the similarity for the architecture between cloud space and traditional client/server structure. Indeed, the communication channel over cloud is based on HTTP request/response approach, mainly because the web app acts like an isolation layer with the hosting cloud system.

C. Middle-Tier Design

The CMPlayer application stores music in the Google App Engine cloud space. The metadata related to the music file are store inside the GAE Datastore, while the music file itself is stored inside the GAE Blobstore. The GAE uses distributed database schema for fast data access and provides SQL like query language for search, add, remove, and replace actions.

Note the Blobstore has an embedded database to describe the data blob. The Google App Engine console provides web-based tools to access the database and facilitate the web app management and maintenance.

IV. DEPLOYMENT AND GAE ADMIN CONSOLE

A. Deployment

While we don't plan to include implementation details in this paper, it is worth to mention the project setup and deployment as all evils are in the details.

The prototype project took the best practice by using Eclipse IDE with plugins for AGE and Android, plus SDKs for J2SE, GAE and Android on Windows XP machine. In addition, we also use the Apache java library for MultipartEntity HTTP POST method support.

To deploy the web app into the GAE cloud space, the developer needs to get a unique application ID from the GAE and apply it to the web app, and then click the "Deploy" button from the Eclipse IDE. The Eclipse will clean up the project and recompile it before delivering the output folder to the cloud.

B. GAE Admin Console

The Google App Engine offers admin console in both cloud space and local development environment. Once the web app is deployed to the Google App Engine cloud space, developer can take advantage of the free out of box admin console to set up the server end application configures, permissions, and security settings, to monitor the cloud resource quota allocation usage statistics and log dump, and to manage the database with Datastore and Blobstore viewers. The default application domain is assigned as "http://cmplayerserver.appspot.com/" for our prototype with *AppSpot* signature inside.

V. TESTING, BENCHMARK, AND PERFORMANCE

The cloud computing environment performance models can be divided into three categories: Client-Oriented, Cloud-Oriented, and Hybrid. [7][8]

We adopted the third approach to construct the CMPlayer application performance benchmark model. Since the music streaming speed cannot be used to quantitatively test the application performance due to the bit-rate control from the media player, we used a single song upload (3.9M Bytes) process and the thereafter playlist download (5.6K Bytes) process to model the performance measurement. To facilitate the test, we added a log dump to both client end and server end applications to get the timestamps for process task start and complete. The tests were conducted in both Wi-Fi network and 3G network environments respectively for performance comparison.

Table 1: Performance - Test Data

		Client (ms)	Server (ms)
Upload (3.9 MB)	**WiFi**	10119	54
	3G	326470	179
Download (5.6 KB)	**WiFi**	190	17
	3G	1410	17

The above table captures the test results in milliseconds for the time spent on uploading and downloading processes over WiFi and 3G networks.

Table 2: Performance – Client vs. Server

		Client (times)	Server(times)
Upload (3.9 MB)	**WiFi**	187	1
	3G	1824	1
Download (5.6 KB)	**WiFi**	11	1
	3G	83	1

It becomes clearer in Table 2 if we change the unit to "times" for the same 4 different processes. The time spent on the client side for the uploading process over the 3G network is 1824 times greater than the time spent on the server side. The time spent on the client side for the downloading process over the WiFi network is 11 times greater than the time spent on the server side.

Clearly, the server side action is fast enough to serve the client's request. The CPU and disk I/O consumed in the cloud space does not compensate the latency on the client end which is caused by the network bandwidth and stability.

The uploading speed over WiFi was 32 times faster than the speed over 3G. The downloading process was only 7 times faster. The result indicates that there was a different bandwidth allocation policy for the WiFi and 3G networks, and the data plan discouraged data uploading task.

VI. ADVANCES AND GLITCHES

Even though Cloud Computing has been evolved for a couple of years, there are still fields need to be improved, especially in the Mobile Software segment, as the available APIs are still mainly targeting on desktop web based application clients. For example, Google App Engine, as a PaaS, provides simplicity but does not support full J2EE API spec and the HTTP request/response control flow seems hard to fit well with Android native applications; While Amazon AWS, as an IaaS, is more flexible but needs considerable time and efforts for the server maintenance and tune-up, and does not eliminate the initial setup const. Another concern is the "lock" effect - porting from one platform to another is not a trial task as there is no common API standard adopted by various cloud environments, and the big players have little stimulation to remove the preparatory barrier. We do notice that RESTful API [9] is promising, but due to the time constrain and/or API maturity, we haven't made a lot of progress on Android platform for our prototype; HTML5 seems to be another promising technique, but if you have a powerful native Java-based Application Framework support, why would anyone build a web-based application? Before WiFi or 4G becomes ubiquity, cloud computing applications would be seriously limited by the network stability and latency.

During the prototype development, we were benefited from the free "out of box" GAE admin console - it reduced the server side maintenance work. We also took advantage of the free quota of the GAE cloud resources (CPU and storage) throughout the entire development process and ended up with zero cost - this PaaS platform does helped us in making the best-ever cost saving on server holding. We want to point out that the streaming capability of the Blobstore API and the Jetty servlet container really speeded up the web app development which is unlikely with AWS.

While we were focusing on the technical details with the prototype modeling and implementation, we do look at the mobile cloud impact to business model and management process. There is no doubt that mobile cloud computing opens a wide door enabling innovations, while the successful application seems still rare with the exception of Salesforce's PaaS platform when it comes with suitable application use cases. Our research is limited but it is interested to follow the industry evolvement and, watch how the Google and Amazon "Cloud Player" applications impact our life style (at this editing time).

VII. CONCLUSIONS

Disregard of the debate whether Software Engineering is an art or a science, or a best practice [1], this paper presents an experimental approach focusing on technical advances and dabbling into intersection of these two hot areas [2]. It is not a surprise that we have seen some glitches during integration, and we may raise more questions than what we could answer, but as long as both Mobile Software and Cloud Computing are heating up, we expect more advances in this industry microenvironment. Through out the CMPlayer prototype development, we get acknowledged about the state of the art of the intersection of these two software computing segments with first hand concrete feeling about the industry pulse.

REFERENCES

[1] Bourque, P.; Dupuis, R.; Abran, A.; Moore, J.W.; Tripp, L.; , "The guide to the Software Engineering Body of Knowledge," *Software, IEEE* , vol.16, no.6, pp.35-44, Nov/Dec 1999, doi:10.1109/52.805471 URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=805471&isnumber=17458

[2] Christy, P. Gartner Identifies the Top 10 Strategic Technologies for 2010. Gartner Symposium/ITxpo, October 18-22, in Orlando. Retrieved November 20, 2010, from http://www.gartner.com/it/page.jsp?id=1210613

[3] WiKi Analysis, Cloud Computing. Wiki Invest. Retrieved November 20, 2010, from http://www.wikinvest.com/concept/Cloud_Computing

[4] Perez, S. Why Cloud Computing is the Future of Mobile. (2009, August 4). Retrieved July 10, 2010, from http://www.readwriteweb.com/archives/why_cloud_computing_is_the_future_of_mobile.php

[5] Fabrizio, C. Five Reasons To Care About Mobile Cloud Computing, International Free and Open Source Software Law Review, Vol 1, No 2 (2009). Retrieved November 20, 2010, from http://www.ifosslr.org/ifosslr/article/view/24/47

[6] Kovachev, Dejan; Renzel, Dominik; Klamma, Ralf; Cao, Yiwei; , "Mobile Community Cloud Computing: Emerges and Evolves," *Mobile Data Management (MDM), 2010 Eleventh International Conference on* , vol., no., pp.393-395, 23-26 May 2010 doi:10.1109/MDM.2010.78 URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5489731&isnumber=5489227

[7] David, L. How to gauge cloud computing performance. (2010, January 28). Retrieved November 16, 2010, from http://www.infoworld.com/d/cloud-computing/how-gauge-cloud-computing-performance-722

[8] Cloud Harmony. Disk IO Benchmarking in the Cloud. Retrieved November 20, 2010, from http://blog.cloudharmony.com/2010/06/disk-io-benchmarking-in-cloud.html

[9] eFreedom. Android: restful API service. Retrieved November 20, 2010, from http://efreedom.com/Question/1-3197335/Android-Restful-API-Service