cAppCloud: Contextual Personalized Application Cloud

Injung Kim
Computer Science
Columbia University
NY, U.S.A.
injungkim@caa.columbia.edu

Sambit Sahu
Cloud and Big Data Analytics
IBM T.J. Watson Research Center
NY, U.S.A.
sambits@us.ibm.com

***Abstract*— Extending ubiquitous data access to personalized applications stack, we propose and build a cloud based intelligent personalized application leveraging cloud to allow users access to their personalized applications from anywhere on any device. Our system is smart and intelligent in the sense that it adapts itself based on (i) available bandwidth between the user location and the cloud system, (ii) user profile that is automatically sensed and learned, (iii) user locations among other context profiles.**

First, we develop a framework for application virtualization that decouples an application from any specific platform or device and enables access to personalized application access at per user level. Our solution is context aware and learns the usage context based on the location and user profile and leverages this to minimize the download bandwidth requirement. Next we provide an implementation of this framework for applications on Windows platform leveraging Amazon S3 cloud storage-although cAppCloud can be implemented for any platform with any other cloud systems. Given the era of Internet of Things (IoT) and various Cloud Enabled Intelligent Applications, we feel that cAppCloud can meet various key requirements to facilitate the interesting scenarios.

Keywords— cloud, personalized application desktop, application virtualization

I. INTRODUCTION

Cloud computing with its resource level virtualization supports on-demand and elastic access to IT computes resources. Such ease of elastic and on-demand resource access has spurred a wide variety of enterprise grade applications that were otherwise very difficult to provide without Cloud based resource. One such application category is the ubiquitous access to data anytime anywhere from any platform or device. Example services include Dropbox [1], OneDrive [2], iCloud [3], and Google Drive [4]. Cloud storage is leveraged by these services to allow users access to their data in a truly ubiquitous manner.

The focus of our paper is to extend this ubiquitous data access to ubiquitous application access for users so that they can seamlessly access their customized applications from anywhere on any type of platforms and devices. We propose cAppCloud - Personalized Application Desktop - that leverages cloud to allow users access to their applications ubiquitously as mentioned above. The key component of cAppCloud is an application virtualization framework that allows cAppCloud to enable platform independent access to personalized applications.

The application virtualization framework consists of two components, i.e., (i) ability to support platform independent access to applications, and (ii) ability to support user specified application personalization in an application independent manner. While the first requirement is relatively easier and feasible to support with simple modifications of existing capabilities such as AppStore [5] and Google Play [6], the second capability requires one to automatically translate the user initiated personalization of an application in one platform to other platforms - which is the core of our proposed cAppCloud application virtualization framework. The implementation of our cAppCloud framework consists of two key components: one is to package all the application and to deploy the applications seamlessly on any computer, and the other is to compose the user profile.

In addition, our solution is context aware and learns the usage context based on the location and user profile and leverages this user profile to minimize the download bandwidth requirement to build the personalized desktop as quickly as possible. This is quite important in a non-enterprise scenario where the bandwidth between the user and the backend cloud may be limited and that the use may be using a few set of applications. For this study, we define "home, work, travel, café/Mall" as the location context to capture different usage types.

While AppStore [5] with iCloud and Google Play[6] with Google Cloud could be leveraged to allow ubiquitous application access, these are neither platform agnostic nor support per user application customization. Desktop cloud is significantly different in that user is provided access to a virtualized platform with remote access to a set of application through remote desktop. We provide access to applications on the device natively that has significant advantage both in terms of performance and functionality.

In order to illustrate the feasibility of our approach, we provide a prototype implementation for Windows based platform with the right level of abstraction leveraging Amazon S3 storage service [7]. Windows registry is a special mechanism to maintain application and desktop configurations, and our prototype implementation also controls the registries to support seamless application and user profile. We demonstrate our prototype by comparing device based and cloud based

978-1-4799-4093-6/14 $31.00 © 2014 IEEE
DOI 10.1109/CloudCom.2014.140

Fig. 1. Users might choose applications they want to use on any platform (left). Each user could execute the applications with their user profile (middle). As soon as the user is disconnected to the cAppCloud framework, the public computer will restore to the original status (right)

compositions of cAppCloud framework. In addition, we discuss areas of future research to extend the cAppCloud framework to the diverse platforms and devices.

In Section 2 we motivate cAppCloud capability and illustrate usage scenarios. Section 3 describes the cAppCloud framework with application virtualization and functional architecture. Section 4 describes how intelligent cAppCloud framework is and Section 5 describes our end-to-end prototype for Windows based platform. Section 6 summarizes our current capability and discusses the future direction to truly achieve application level ubiquity leveraging cloud.

II. Motivation and Usage Scenario

In this section, we describe cAppCloud - a ubiquitous personalized application desktop cloud that allows users to access their applications from anywhere on any platform. First, we motivate our proposed solution and compare it with related work. Next we illustrate cAppCloud usage and its enabling key elements.

A. Motivation

In a truly emerging ubiquitous computing era, one should be able to access ones personalized application besides the data from anywhere on any platform. Cloud storage based services such as Amazon S3 [7], Google Drive [4], Dropbox [1], or Microsoft OneDrive [2] allows ubiquitous data access. However, these do not consider how access to ones personalized applications.

One may argue that a user may access to his/her applications remotely. For example, using remote desktop or cloud server has clear disadvantages which are network dependency, incompatibility with some operating systems, downtime, or bottlenecks. Many usage scenarios and applications would not meet the desired performance or the constraints.

While services like iCloud [3] a step in this direction that could be leveraged to build the capability that being advocated by our cAppCloud solution, it is not platform independent and also does not support application personalization. Our previous work [8] supports the portability of application and user profile level based on the USB hard drives, but it has limited storage capacity, as well as exposes itself to the security threat like a lost device, and could only deploy platform dependent applications.

Our proposed solution allows one to access personalized applications on any platform - a capability that extends the ubiquitous data access. cAppCloud allows users access to their personalized applications thereby providing a seamless connected experience in a platform independent manner.

B. cAppCloud Usage Phases

Let us illustrate how cAppCloud facilitates the capability stated above. Figure 1 left image illustrates the initialization phase, i.e., the very first time a user accesses an application using cAppCloud from a device. This phase is similar to installing and accessing an application from AppStore [5] for iOS, Google Play [6] for Android platform. But cAppCloud supports this in a platform agnostic manner by automatically selecting appropriate application version and installation process. The user is then allowed to use these applications with ubiquitous data access. The user may choose to personalize the applications thereby changing application configurations.

Figure 1 middle image illustrates the capability to personalize and customize the applications. For example, one may choose to change ones screen wallpaper. It is feasible as cAppCloud framework maintains application customization by keeping track of changes to appropriate configuration files. Thus next time a user accesses the same application, the saved configuration files are used instead of initial configuration files.

Figure 1 right image illustrates the cleanup phase that is activated as soon as user wants to end the session. This is a critical phase for privacy concern as user could be accessing his/her personalized applications along with data from a public computer.

As illustrated, users may easily access their data and any application with a simple connection to the cAppCloud framework that is enabled on top of cloud storage. As cAppCloud restores applications using saved per application configurations in a platform independent manner, users are provided a truly ubiquitous application level access.

III. Framework and Architecture

In order to support ubiquitous application access, cAppCloud provides two key enabling components: (i) platform and OS independent application access, (ii) application personalization. These are the core capabilities that cAppCloud brings through application virtualization framework that we propose and provide an implementation.

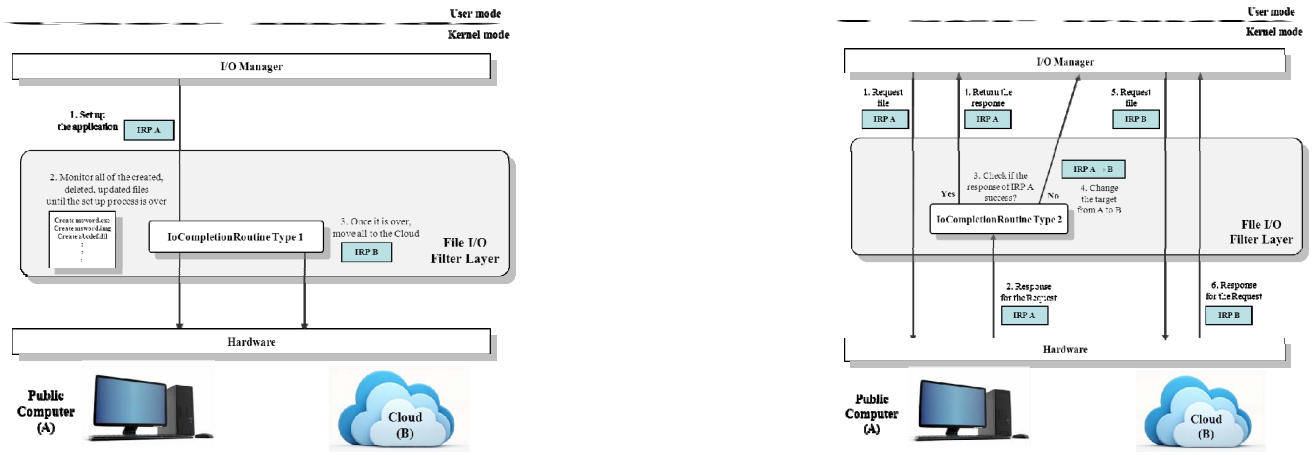


Fig. 2. Packaging all the applications (left). Deploying applications seamlessly (right).

A. *Application Virtualization Framework*

Supporting the first requirement is relatively easier. cAppCloud needs to detect the platform and OS level details of the device that the user logs in to access cAppCloud. Based on the platform details, it chooses the right application installation files and installer.

The second requirement of application personalization is tricky. cAppCloud needs to capture the changes the user makes to the application configuration on the current platform and then derive the equivalent changes for the other platforms.

These two requirements are achieved by describing an application that captures platform dependent installation and configuration relationships across the platform. These together are our framework for application virtualization. A suggested implementation is a configuration specification of application defined in a manner that allows one to link configuration elements through a relationship. Once an element is changed in one platform configuration, through this relationship, this change is propagated to other platform configurations. In some sense this is similar to what an OVF format is for hypervisor independent virtual machine specification.

B. *Functional Architecture*

Next we describe the functional aspect of the architecture. It could be divided into (i) preparing stage, (ii) deployment stage and (iii) composition phase.

1) Preparing Stage (Packaging All The Applications): The cAppCloud framework lets the users make the application packages they want to carry, and to make the application package, they need to have its general set up file. Figure 2 left image shows the key steps to make an application package. For example in the case of Windows platform, all the file actions are requested by I/O request packet (IRP) on the kernel level. Since it works on the kernel level, we call our functional module as a file I/O filter layer, and the IoCompletionRoutine type1 function on the file I/O filter layer performs to make the application package.

For the first step, through the cAppCloud user interface, they may choose any application's set up file, and then start it to let the IoCompletionRoutine function know which process is to be monitored. As soon as the set up process is started, the function also starts to monitor all the actions such as create, update, and delete files. And then, once the set up process is done, the function detects the end of the set up process and moves all the changed or created files to the cloud storage with file path, name, version, and platform information. In the Figure 2 left image, we mark an original file request as IRP A which target is a public computer, and mark a copy request of application files as IRP B which target is cloud storage.

The point is these packaged applications cannot be executed on other computers because all the application processes tries to find their files on the initial computer not on the cloud storage. Therefore, following stage will describe how to deploy application packages seamlessly.

2) Deploying Stage (Deploying Applications Seamlessly): Between two different stages; preparing stage and deploying stage, there is no need to work on the same public computer but need to connect to the user's cloud storage space. Once the user makes the application packages into the cloud storage, he or she can move all around the world, and then have a sit in front of any platform to keep going on his/her work.

Since we suppose that all of the application packages are already stored on the cloud storage by the preparing stage, there could be all the versions of diverse applications for all kinds of platform on the cloud storage. Due to the benefit of the cloud storage, there is no storage capacity limitation and the user may easily put any application package and their large data, either. However, the application package could not be executed itself because all the related application files do not exist on the public computer but on the cloud storage. Figure 2 right image explains how to redirect all the file requests from the public computer to the cloud storage. In this case, IoCompletionRoutine function on the file I/O filter layer performs to let the application package work seamlessly.

The concept is that the requests of the application will fail since the file does not exist on the public computer. Therefore,

before the failed response is returned to the application itself in the user mode, the IoCompletionRoutine function captures the failed response, changes its target from the public computer to the cloud storage, and makes the request issue again. Then, I/O manager issues the request to the cloud storage without sending the failed response to the application, and the new response will be returned successfully. It is also possible that the IoCompetionRoutine bypasses the successful response when the requested file exists on the public computer. In the Figure 2 right image, we mark an original file request as IRP A which target is a public computer, and mark a redirected request of application files as IRP B which target is cloud storage.

With the deploying stage, when the user connects the public computer to the cAppCloud framework on the cloud storage, the applications stored on the cloud storage be executed on any public computer.

3) Composition (User Profile Automation): Unlike application part of the cAppCloud framework, composition of the user profile is intuitive. The user may choose any user profile they want to carry and store it into the cloud storage through the cAppCloud framework. When the cAppCloud framework is running, it will backup the user profile of the public computer, then change it to the user's one which is already stored on the cloud storage. For example, if a user stores his/her preferred wallpaper, screensaver, favorite links, or even internet cookies into the cloud storage, when the user is connected to the cloud storage from the public computer, the cAppCloud framework will automatically compose the user profile into the public computer. Lastly, there is a cleanup process which restores the original user profile into the public computer right after the connection is dropped. The cleanup process also removes the file I/O filter layer from the public computer. With these automatic composing user profiles,the user could work on the personalized application desktop cloud.

IV. CONTEXT AWARE CAPPCLOUD

The previous section described the system architecture that did not assume any bandwidth constraint. It assumes that sufficient bandwidth is available between the cloud system and the user device so that all the application images can be downloaded onto the user device in real time to create the personalized desktop for the user. However there may not be sufficient bandwidth available to download all the applications especially in the wide area network – which is typically the case in an non-enterprise scenario. We illustrate later in the section using several Internet measurements that based on usage contexts, the available bandwidth may be severely limited.

In this section, we propose an intelligent approach that accounts for bandwidth limitations and adapts the solution in a context aware manner accounting for a variety of factor to optimize the user experience. When the bandwidth is limited, it is always not feasible to download the entire desktop content. Also it is always not required to download the entire set of applications. Consider the following scenario: When a user is at home, the applications and data usage may be different compared to when at work. Also it may very well be different when one logs in from a café or when one is travelling or

```
<UserProfile id="ijkim">
     <Desc>ijkim's user profile</Desc>
     <AppProfile id="ijkim_office">
          <Platform>Personal Computer</Platform>
          <OS>Ubuntu 14.04 64bits</OS>
          <Bandwidth>12.7Mbps</Bandwidth>
     </AppProfile>
     <AppProfile id="ijkim_home">
          <Platform>Personal Computer</Platform>
          <OS>Microsoft Windows XP SP3</OS>
          <Bandwidth>20.382Mbps</Bandwidth>
     </AppProfile>
     <AppProfile id="ijkim_phone">
          <Platform>Mobile Phone</Platform>
          <OS>Android 4.4.4</OS>
          <Bandwidth>4.57Mbps</Bandwidth>
     </AppProfile>
</UserProfile>
```

Fig. 3. User profile has various kinds of workspace

```
<AppProfile id="ijkim_office">
     <Desc>ijkim's application profile at office</Desc>
     <Application>
          <Name>Adobe Acrobat Reader</Name>
          <Version>9.4.6</Version>
          <TotlaSize>118MB</TotalSize>
          <LastAccessTime>17:30 15/08/2014</LastAccessTime>
     </Application>
     <Application>
          <Name>Google Chrome</Name>
          <Version>36.0.1985</Version>
          <TotlaSize>450MB</TotalSize>
          <LastAccessTime>14:15 15/08/2014</LastAccessTime>
     </Application>
</AppProfile>
```

Fig. 4. Application profile has several application lists with its total size and last access time to decide which application prefetches first

checking some web sites from a Mall. Thus the usage of applications and the dependent data are very context specific. It may be possible to learn this context and be possible to just fetch only a subset of the applications and dataset instead of the entire dataset – which would be quite useful for bandwidth limited case.

Before defining the algorithm in detail, first we provide an intuitive sketch of the algorithm. The key idea behind the algorithm is based on "Contextual User Profile" that we maintain and learn for each user. Each user has a predefined set of user context. While it is not limited to these set of predefined states, we explore with these set of predefined labels: Home, Work, Travel, Café/Mall. These are location based activities that we feel define the characteristics of application usage. For each user, based on where a user may be logging in from, those application sets would be downloaded instead of full set. This metadata would be learned from the past usage just like any other caching based approach. This metadata would be maintained as a per user profile in the backend in the cloud as part of our system.

Both the user profile and application profile are designed

for the various kinds of personalized workspace. By logging the user's various workspace and application preference into the user profile and application profile, personalized workspace could be composed depending on where the user is and which application the user prefer to use. Also, prefetching applications consider to network dependency because if there are much streaming requests, the response time for the application will be slow. Therefore, after checking the network situation of the current platform that a user is using, perfecting most recently accessed applications will improve the entire application streaming. We also mention an algorithm how the application prefetcher decides which application prefetches first after learning which applications the user prefers to use on which workspace by having the last access time of the application that the user actually requests.

1) User profile: Our current implementation is applicable to store desktop data such as wallpaper, screensaver, favorite links, or even internet cookies into the cloud storage. However, the user may use different applications as well as desktop data on their circumstance (e.g., a user may mostly use web-browser for online shopping at home, but he/she may use document applications at work.). Figure 3 shows the user profile has various kinds of workspace such as office, home, and even mobile phone. In order to specify the diverse platform for the user, the user profile has platform name, OS type, and bandwidth of each platform. With this user profile, all of the applications could support for the diverse platform for a user.

2) Application profile: The fact that provisioning in our current implementation deploys applications when a user actually requests. In practice, however, prefetching applications before a user requests may be desirable to decrease network dependency, but this clearly requires prefetching by figuring out which application is better to prefetch first. Therefore, Figure 4 provides that application profile has the application information such as its name, version, total size and last access time. Especially total size and last access time of the application are to decide which application prefetches first. In case the user may not request the application prefetched, the last access time will not be updated. Therefore, the last access time means the time when the user really uses the application.

3) Algorithm of application prefetcher: Figure 5 shows the algorithm how the application prefetcher will prefetch the applications and manages user profile and application profile. U is a user, A is an application, and AP is an application profile. For the first, a user chooses a user profile that the user wants to compromise the specific applications into the current platform. If the user wants to create application preference for the new platform or the user profile is not up-to-date, the user will update a user profile and the application prefetcher learn the user profile preference. Once the user chooses or adds the proper user profile, the application prefetcher gets the preferred application to prefetch by checking for the last accessed time of the application. Then, it will check the total size of the application and the network bandwidth of the current platform, then figures out how long prefetching the

Assumptions:

ChooseUP() chooses proper user profile
UpdateUP()/UpdateAP() updates the user profile/application profile
GetPreferredApp() gets the preferred application to prefetch
CheckPrefetch() checks whether the application will prefetch or not
PrefetchApp() prefetches the application
LearnUP()/LearnAP() learns user/application preference

Algorithm:

```
U = ChooseUP()
IF U != uptodate THEN
    UpdateUP(U)
    LearnUP(U)
ELSE
    A = GetPreferredApp(AP)
    WHILE CheckPrefetch(A.totalsize, U.bandwidth)
        IF A == requested THEN
            UpdateAP(AP)
        ENDIF
        PrefetchApp(A)
    ENDWHILE
    LearnAP(A)
ELSEIF
```

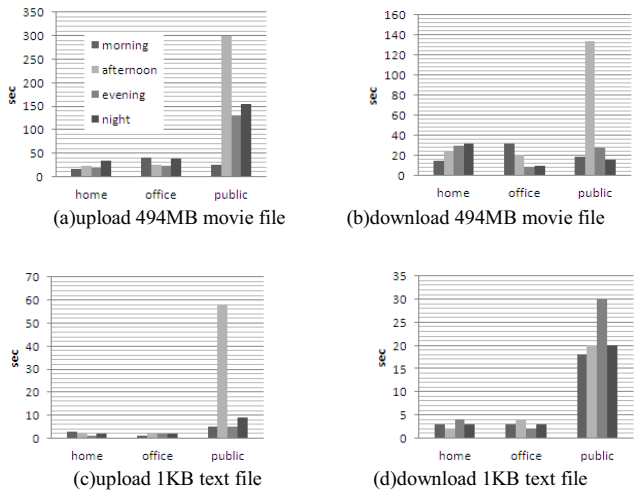
Fig. 5. Application prefetcher will prefetch the applications and manage user profile and application profile.

application will take. In case the user actually tries to launch any application, the application prefetcher updates the application profile with the current access time. Also, while the bandwidth is enough to prefetch, it will prefetch all of the applications in the order of most recently accessed application. In this algorithm, there are learning functions to have the user's preference about the platform and application. The best scenario of this algorithm is that the application prefetcher prefetches all of the applications before the user wants to use since the network bandwidth is enough and the prefetcher's prediction is correct for all of the prefecthes. Therefore, the user may launch the applications without network latency since all of the applications are already prefetched on the platform.

V. PROTOTYPE IMPLEMENTATION

This section shows a prototype on the Windows platform leveraging Amazon S3, and then presents performance metrics. This paper presents an implementation of the cAppCloud framework for applications and user profile on Windows platform leveraging Amazons S3 cloud storage [7]. The experimental test-base is the Intel dual core 2.6GHz*2, 2.93GB RAM, Microsoft Windows XP SP3, with 144.0Mbps WIFI network. We choose the Amazon S3 storage in Tokyo, Japan as the nearest cloud storage from Korea where the test was performed. To recognize the cloud storage as a typical disk partition, CloudBerry Drive 1.3.0 lets the user map a network drive to cloud services and use remote files like they right on the computer.

In view of wireless connection, diverse wireless environment need to be considered as well. In a broad or abstract sense, the scenarios for the use of wireless network may be categorized as home, office, and public environment. From the diverse environments, there exist general



(a)upload 494MB movie file (b)download 494MB movie file

(c)upload 1KB text file (d)download 1KB text file

Fig. 6. Comparison of file transfer time on the several wireless environments

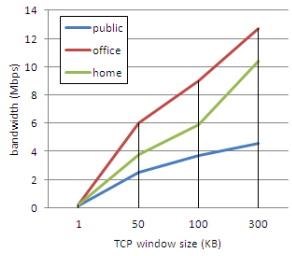


Fig. 7. TCP bandwidth measurement on three different environments

problems on the wireless network. There could be too many users trying to use the same channel, radio frequency might interfere with WLANs, and wireless adapter settings such as QoS would influence the use of wireless network. Therefore, Figure 7 shows the comparison of file transfer time on the several wireless environments we may face with. We examine the upload and download time of different size of files on several time slots; morning, afternoon, evening, and night. The experiment at the public place has from 5 to 54.0Mbps WIFI network, and then office and home have relatively high WIFI throughput, 144.0Mbps and 130.0Mbps, respectively. Each experiment has been tested 10 times on three different days. The speed to upload and download in public environment has generally worse than others because there are many users using the same channel at the public place and noise such as radio frequency is out there. The reason why there are wide variations of the throughput in public environment is because it mostly gives free WIFI service with a limited quality of service (QoS) of the wireless adapter settings. On the other hands, office and home have relatively constant throughput because it set up the wireless network with a password to serve it exclusively.

To check each bandwidth of three different environment; public, office, and home, we also performed additional experiments to consider how the bandwidth is diverse on the different WLAN environments. We have used an iperf 2.0.5-2 tool to measure the network performance, and repeated the above experiment for 100 times for the TCP bandwidth on three environments with different window size in Figure 8. From these experiments we see that there is a bandwidth limitation for the public environment. The graph also describes that the bandwidth is getting higher for the bigger TCP window size, however, there could still exist the defeats of wireless network such as concurrent connection, noise interference, etc.

In general, we see that using cloud storage to store personalized application and data definitely takes longer than device based or public computer itself. Also, when the user uses the wireless network, its bandwidth will be an important fact to determine the entire performance of our framework. Unlike LAN environment, the wireless network environment needs to be considered how many applications the user would be better to prefetch, and what size of applications the network environment could deal with. Therefore, through the user and application profiles which were created and managed by our algorithm, applications is better to prefetch, particularly when the network bandwidth is enough to prefetch.

VI. CONCLUSION

In this paper, we extend the ubiquitous data access to ubiquitous personalized application access leveraging cloud storage. Towards facilitating access to personalized access to applications natively from anywhere on any device, we proposed and built cAppCloud - a personalized application desktop leveraging cloud. First, we developed a framework for application virtualization that decouples an application from any specific platform or device and enables access to personalized application access at per user level. In addition, we built context aware user profile to only prefect part of the application set to minimize the bandwidth requirement. The system learns user behavior and maintains a location and usage aware context in a user profile and leverages intelligently to build the personalized desktop as quickly as possible.

Next we provide an implementation of this framework for applications on Windows platform leveraging Amazon S3 cloud storage using application virtualization framework. Note that this same implementation framework can be extended to Linux, Mac OS, Android based platforms - which is in our future implantation roadmap.

Future work for wider supports of large-scale ubiquitous personalized application desktop cloud such as other operating systems, other cloud storages, or even other client machine like a smartphone is needed. It is also possible to improve the performance when the framework works on the remote storage by using cache or making full use of local applications first not to have much streaming requests.

REFERENCES

[1] Dropbox, http://www.dropbox.com
[2] Microsoft OneDrive, https://onedrive.live.com/
[3] Apple iCloud, https://www.icloud.com
[4] Google Drive, http://drive.google.com
[5] Apple AppStore, http://store.apple.com
[6] Google Play, https://play.google.com/store
[7] Amazon Simple Storage Service(S3), http://aws.amazon.com/s3
[8] I. Kim, M. Hwang, W. Lee, C. Park, "u-PC: personal workspace on a portable storage", Proceedings of the 4th International Conference on Mobile Technology, Applications, and Systems, pp. 220-225, 2007