Smile: Streaming Management of Applications and Data for Mobile Terminals

Yangyang Zhao

Institute of High Performance Computing,
Department of Computer Science and Technology,
Tsinghua University, Beijing 100084, China
E-mail: zhaoyy09@mails.tsinghua.edu.cn

Wentao Han

Institute of High Performance Computing,
Department of Computer Science and Technology,
Tsinghua University, Beijing 100084, China
E-mail: hwt04@mails.tsinghua.edu.cn

Ruini Xue

School of Computer Science and Engineering,
University of Electronic Science and Technology of China,
Chengdu, 611731, China
E-mail: xueruini@uestc.edu.cn

Wenguang Chen*

Institute of High Performance Computing,
Department of Computer Science and Technology,
Tsinghua University, Beijing 100084, China
E-mail: cwg@tsinghua.edu.cn
*Corresponding author

Abstract: With the rapid growth of the mobile phone industry in recent years, consumer habits in using mobile applications have changed significantly. Mobile clients have replaced desktop computers as primary internet access devices. However, mobile phones have limited battery life, low processing power, and limited storage capacities. As mobile devices are easily lost or damaged, better data management schemes are also required. In this paper, a streamed application and data management system based on Transparent Computing technology is proposed to support more secure, better managed mobile phones. Experimental results show that the proposed system is a feasible and efficient solution for future mobile computing applications.

Keywords: Transparent Computing; Application Streaming; Data Synchronization; Cache Eviction

Reference to this paper should be made as follows: Zhao Y., Han W., Xue R. and Chen W. (xxxx) 'SMILE: Streaming Management of Applications and Data for Mobile Terminals', *Int. J. Cloud Computing*, Vol. x, No. x, pp.xxx–xxx.

Biographical notes: Yangyang Zhao received his bachelor in computer science and technology in 2009 from Tsinghua University, China. He is now studying in department of computer science and technology in Tsinghua University for a master degree. His research interests include transparent computing, mobile cloud and pervasive computing.

Wentao Han received his bachelor degree in computer science from Tsinghua University in 2008. He is now a PhD student in the Department of Computer Science and Technology, Tsinghua University. His research interest is on operating systems, distributed systems and mobile computing.

Ruini Xue received his PhD in Computer Science in 2009 from Tsinghua University, China. Currently he is a lecturer at the School of Computer Science and Engineering in University of Electronic Science and Technology of China. His research interests includes cloud computing, distributed systems, and mobile systems.

Wenguang Chen received the B.S. and PhD degrees in computer science from Tsinghua University in 1995 and 2000 respectively. He was the CTO of Opportunity International Inc. from 2000-2002. Since January 2003, he joined Tsinghua University. He is now a professor and associate head in Department of Computer Science and Technology, Tsinghua University. His research interest is in parallel and distributed computing, programming model and mobile cloud computing.

1 Introduction

The growing popularity of devices like smartphones and tablets is attributed to portability, and the availability of tens of thousands of mobile applications (Apple 2011*a*, Google 2011*a*).

The portability of such mobile devices makes them indispensable for daily use, and through applications, they offer functionality comparable to personal computers. However, application management in mobile terminals is still a challenge for Managing the installed or purchased applications is an inconvenient process. Installing an application on mobile devices is a multi-step process. For example, in Apple devices, this process involves launching the App Store, browsing or searching for the application, pressing install to start the download, waiting for the download to complete, and finally activating the application. To install an update for the application, the user needs to launch the app store again and select `UPDATE` to manually download the updated version. Often, security patches for the operating system (OS) need to be applied. Current security solutions are not suitable for mobile devices. To perform a system upgrade, all applications first need to be backed up and restored with the help of the iTunes computer program. As many users

are unfamiliar with these steps and the available tools, almost 78% of all iPhone users never install new applications (Ligang 2010). Therefore, the application management system needs to be simplified.

If the user loses or replaces a device, he/she would want to restore the previous device environment. For this to be possible, all previously installed applications should be reinstalled automatically on the new device and all user application data must be restored to make the applications function properly. This is also a challenge.

- The current application management model is to install applications locally. If the remaining disk space is insufficient, the user has to remove previously installed applications or delete local files to make space. This situation is common as user generated data and applications share the same storage space in mobile devices. For example in the Android OS, user applications are installed in the `/data/app` folder, and all user-generated data; comprising documents, photos, videos, and songs etc.; are stored in `/data/data`. Therefore, when more data are stored, less space is left for applications. Mobile device users tend to capture a lot of video and photos, which consume a lot of disk space (Poulsen 2011, Purdy 2010).

  Another problem is that if the user wants to use an application, which was just uninstalled to make room for another. The user then has to uninstall another application to reinstall the required application. Thus, the user is caught in an *uninstall-install* loop. In addition, a user who does not know how to uninstall applications cannot use new applications. Thus, the current implementation is neither user friendly, nor resource efficient.

- Managing application data presents a challenge even for users who know how to uninstall and install applications. For example, a user may have configured many options in a note taking application or completed several levels of their favorite game. If these preferences are not restored correctly on the new device, the user might stop using the application. For most current applications, users have to identify the application data themselves, which then needs to be backed up and restored manually. Some tools like iTunes facilitate this process, but only for the most popular applications.

Due to the rapid widespread deployment of wireless networks, especially 3G networks, mobile devices can be always connected and use several networked applications. There is a clear gap between the current application management schemes in mobile terminals and user requirements. For simplicity, we define application state as the application binary and its associated data. The problem then is to manage the application state transparently. The property of transparency here denotes the following: (1) User installed applications are always available and directly accessible through clickable shortcuts. Users do not have to be concerned about the location used for data storage; (2) In most cases, users should be able to open an application and use it instantly; (3) Users should not detect any difference between the current and previous environments after switching to a new device.

To address these problems, we propose a method called SMILE(***S**treaming **M**anagement of Applications and Data for Mob**ile** Terminals*), which adopts the transparent computing paradigm (Zhang et al. 2010, Tian et al. 2009, Zhang and Zhou 2006, 2009). SMILE deploys a *subscription model* to manage applications.

Users can subscribe to applications without being limited by the local storage capacity. SMILE always shows all subscribed applications to the user, and updates them automatically. Thus, users no longer need to manually download, install, upgrade, backup, and restore applications. Subscribed applications include both free and purchased applications. As the mobile terminal has limited storage and computing power, it only stores some application states; all states of the users' applications are stored in the remote server. However, the device shows all the applications to the user, making it appear as if they are all installed locally. The device caches a user's most frequently used applications locally, which can be replaced if a user accesses an application that is not stored locally During this process, data associated with the application being replaced will be synchronized with the server for future retrieval. The new application will be downloaded with its associated data automatically. This mitigates the problem of data loss if the device is lost. SMILE streams remote applications to local storage based on user requests, while the physical location of application data is abstracted and hidden from the user. Streaming in SMILE is coarse-grained in terms of the entire application; the application cannot be launched before the streaming finishes. Application data is always transferred along with the application. Therefore, the server always stores the latest data, which can be retrieved on demand for the ease of the user.

It is difficult to make SMILE function transparently. The three challenges involved in ensuring transparency are:

- Seamless Application Scheduling. Subscribed applications may not all be stored locally, but this does not affect the user who is able to see and use all subscribed applications. Current mobile application management systems can only provide access to local applications. In contrast, SMILE provides a unified interface where the user cannot distinguish between local and remote applications. SMILE keeps track of the physical location of applications and can remove old applications, and stream new applications and data automatically.

- Seamless Application Management. Users of SMILE will be able to install, uninstall, and update applications as they are used to doing, regardless of whether applications are stored locally. SMILE offers a mechanism to manage cached or un-cached applications seamlessly. Installation and update operations in SMILE are smoother and better managed.

- Efficiency. If local storage is very limited, and the user often changes applications, SMILE has to reduce the overhead of application replacement to avoid frequent re-installation and to control the volume of data transferred over the network. SMILE is able to devise a simple but effective application replacement strategy by investigating the application usage frequency. The system is based on the Least Recently Used algorithm, but takes application characteristics into consideration, and is better suited for mobile terminals.

The three contributions of this paper are as follows:

- Novel mechanism to manage application states. SMILE differs from current application management models as it streams the entire application and its data. SMILE is also unique in having the ability to access a virtually unlimited

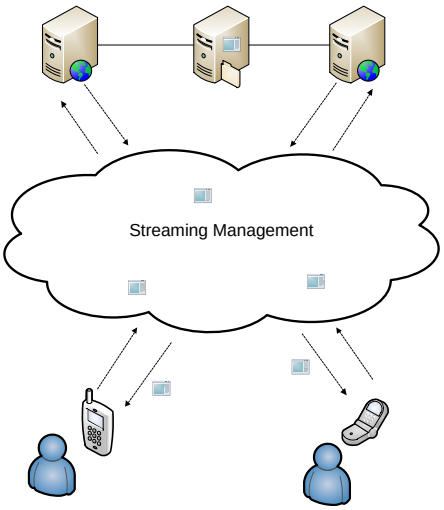

Figure 1 Infrastructure of SMILE.

number of applications from a mobile device, and to restore the complete device environment without user intervention.

- Application caching and replacement strategy. This strategy involves having a higher probability of local caching for frequently used applications to reduce streaming time.

- An implementation of SMILE on Android with extensive evaluation. A prototype of SMILE was implemented on Android to demonstrate the proposed system and extensive experiments were carried out for evaluation. Experimental results show that the performance of SMILE is adequate and it is a feasible solution for future mobile computing devices.

The paper is organized as follows. Section 2 details the principles of application streaming and the system design. In Section 3, application management with SMILE is discussed. Section 4 describes the issues and problems associated with the server. Section 5 details the implementation in Android, which is followed by an evaluation of the system performance in Section 6. Finally, Section 7 summarizes related work while the conclusion forms in Section 8.

2 Design

2.1 Principles of Application Streaming

SMILE offers a new paradigm for mobile applications in a network-connected environment. Several application meta servers store the application data and user application data. Front servers handle requests from mobile clients, while

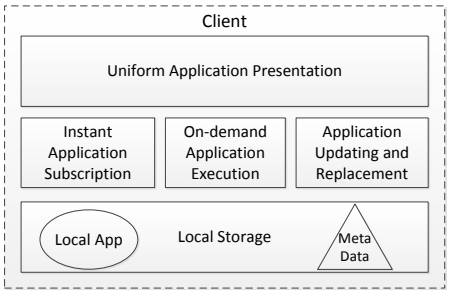

Figure 2 Client Modules.

maintenance servers manage the update and addition of new applications to the meta servers.

Mobile devices like smartphones or tablets connect to the servers over the network. Applications and their data are stored on servers, and the device only holds part of the data as show in Figure 1. All subscribed applications are visible on the mobile device, regardless of where they are stored. Users can launch subscribed applications like local applications. Users do not have to know the storage location of an application, or worry about losing application data. Furthermore, the proposed system automatically updates and replaces subscribed applications as required.

The SMILE system distinguishes itself with the following features:

- Convenient and effective user experience: The subscribe/unsubscribe options in Smile are more convenient and take less time to use than similar options in other systems. In addition, users can access more applications than what the local storage can hold, without any extra effort or complications.
- Transparent application scheduling: Although not all applications are stored locally, users do not need to know the actual storage location, they can continue to access applications as before.

2.2 System Architecture

We designed client and server components for our system. Each client can be a mobile device, and the servers can be regular PCs or servers than runs the required software. The server and client are connected through Wi-Fi or 3G. Users can connect to this system after some basic configuration and account creation.

2.2.1 Client

The client architecture is composed of four parts as shown in Figure 2: the Instant Application Subscription, Uniform Application Presentation Layer, On-demand Application Executor, and the Application Update and Replacement module.

The Instant Application Subscription module controls the installation and removal of applications in the SMILE system. Unlike current app stores or markets, it does not download or remove the application directly to the mobile device. This module only imports/changes minor information (called metadata) into the system and enables convenient trial and removal of applications .

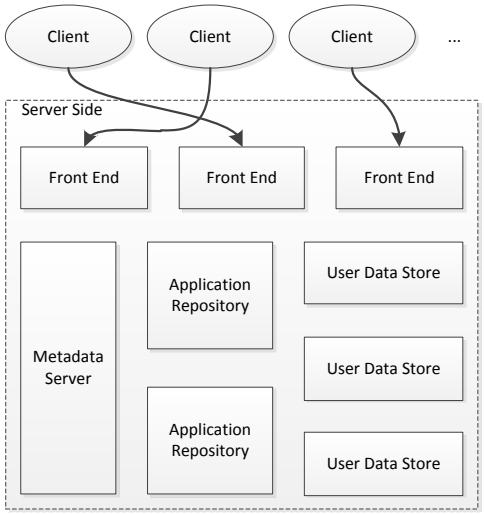

Figure 3 Server-side architecture.

The Uniform Application Presentation module displays all applications associated with the current account on the device uniformly, irrespective of the storage location.

The On-demand Application Execution module controls the execution of applications in SmileThis module is in charge of launching cached applications and the loading of un-cached applications.

The Application Update and Replacement module keeps track of the applications and their status. It contributes to the process of updating, removing, or recycling applications. Through this module, application content and status are seamlessly mirrored on the client and server.

2.2.2 Server

The server side of Transparent Application Streaming is composed of four components, namely, the Front End (FE), the Metadata Server (MDS), the Application Repository (AR), and the User Data Store (UDS), as shown in Figure 3. The Front End communicates with clients; it dispatches requests from the clients to the corresponding components behind the Front End, and sends back responses. The partition layout of applications and user data are stored on the Metadata Server. Application packages (.apk files under Android) are stored in Application Repositories, and user data are stored in User Data Stores. These four components form the server side of Transparent Application Streaming, and together provide services to clients.

The Front End is the only component that the clients directly communicate with. It isolates clients from other server side components. Clients send various kinds of requests to FE, and wait for responses. To optimize performance, FE leverages a caching mechanism and responds immediately if the information requested is entirely cached. Otherwise, it requests the location of the required data from MDS, and then fetches data from the appropriate AR or UDS.

Each application stored in AR has its own identifier (*aid*), which is the package name of the .apk file in this study. Each user of this system has a unique identifier (*uid*), too. Thus, the (*uid*, *aid*) pair identifies the user data for application *aid* belonging to user *uid*. These identifiers are used to partition applications and user data across different ARs and UDSes.

MDS has two main functions: The first is tracking the partition layout of applications and user data. FEs query MDSes to obtain the location of the requested application and the required user data. New applications or user data added to the system are allocated a location by MDS; this location is a specific AR or UDS. The other function of MDS is to administer all server side components. It monitors heartbeats from the FEs, ARs, and UDSes. If a component is lost, MDS alerts the system administrators.

ARs and UDSes are essentially key-value stores. Applications and user data are fetched and stored according to identifiers, either *aid* or (*uid*, *aid*).

3 Client

In this section, we discuss several client side components, including Instant Application Subscription, Uniform Application Presentation, On-demand Application Executing, Application Replacement and Updating strategy.

3.1 Instant Application Subscription

The Instant Application Subscription module acts as the application market for the client. It retrieves the application information list from the FE, and sends back subscribe and unsubscribe instructions.

In a conventional mobile system, subscribing to an application means downloading and installing the application package file and being constrained by space limitations. In comparison, subscribing and unsubscribing to an application with SMILE is relatively painless. Users can subscribe to any application irrespective of size and the application is available to use immediately.

This subscription process is more efficient because only the metadata (explained in Section 3.2) is downloaded, not the whole application.

However, applications will not always just store the metadata locally. In fact, in most cases, the client device will have to store all the application content. Section 3.3 and 3.4 provide details of how an application switches between these two storage options.

Unsubscribing from applications is very similar to conventional mobile environments. All data stored locally is deleted, whether it is just metadata or the complete application, which can be deleted through a regular remove process.

3.2 Uniform Application Presentation

In a conventional mobile device environment, all applications are stored locally either in the ROM or in external storage. In the proposed system, a subscription does not necessarily install the whole application, but the subscribed application needs to be visible to the user. Thus, our application presentation layer needs to be different.

In conventional mobile devices, applications are shown on the home screen, and the home screen itself is an application that queries the package manager for available application packages. Thus, the home screen application arranges the application shortcuts depending on the number of packages available. It retrieves the icon and label of those packages when they have to be displayed. In the proposed system, locally cached applications are displayed in the same way. The challenge is to display non-local applications. An application consists of several components, of which the execution code takes up the most space in the application bundle. The components needed to display the shortcut take up very little space. All these display components are collectively referred to as the metadata.

For a subscribed application, this metadata is stored locally for display. Every component of metadata is displayed on the home screen, and the non-local application can be launched and deleted exactly like a local application. Local applications can be displayed unchanged. Thus, the Uniform Application Presentation module is a combination of the original and metadata presentation modules.

3.3 On-demand Application Execution

The last section suggests that an application's status can be cached or un-cached, where cached applications are stored just like applications in current mobile environments, whereas un-cached applications only store metadata locally. To create a seamless experience, a special execution module is needed to launch cached and un-cached applications.

A mobile application in conventional mobile environments is executed as follows:

- User to system action transportation: The home screen will store the application presentation data (label, icon) and corresponding package information (package name, storage position, etc.) in its memory. When the user taps an application icon from the home screen, the package information of the destination application is found by calculating the coordinates of the tap event. A system action with the application entry point and launch parameters is then performed.
- Application data loading: The system action launches the appropriate application. The application data will be loaded into memory and the application will execute.

Application execution in the Smile system occurs as follows:

- User action to system action transportation: The Smile system home screen also first creates a system action for each application launched. However, the system action for un-cached applications is formed differently, as a special action calling for a remote load process, which will be used in next step.
- Application data loading: The loading process for cached applications is again similar to conventional devices. For un-cached applications, the information in the action will be converted to a remote address for a loading service, implemented as a system service to retrieve the application content from the meta server. Once the data is fully retrieved, it is installed like a normal application. The rest of the process is the same as for local applications.

- Application post launch action: This step is only required for un-cached applications. After an un-cached application is downloaded, it will become a cached application. The home screen will update its data so that the system action for these applications is not an action call for a remote load process. Subsequent application calls will happen exactly like cached applications.

The proposed system shares many steps with the conventional approach. However, the loading process has a delay between the subscription stage and the execution stage. We also have different actions for different type of applications, and the binding of applications and actions is not fixed.

3.4 Application Replacement Strategy

The previous section described the execution of cached and un-cached applications. If the user subscribes to too many applications, he/she cannot all be cached locally and we need to decide which ones to cache locally. When there is not enough space for a new application, some cached application will have to be replaced.

The replacement strategy is an important component of our system as there is no limit on the number of subscriptions available to a user. Our goal is to allow users to try out several applications even with limited storage space, which having a low penalty in terms of waiting time and network traffic cost.

First, we define the granularity of replacement. Like the replacement policy used by CPUs to deal with the cache lines, we can opt for a coarse grain or a fine grain. For the coarsest grain, we may replace all the applications in one operation, while for the fine grain we would replace only one or a small number of applications. Under a coarse grain policy, if a new application needs to be cached, we would replace several applications with new applications. We would be doing some prefetch work in the process, with the expectation that the user will use those prefetched applications. However, unlike a memory cache, the usage pattern of applications is irregular and unpredictable, although we can gather information on the usage frequency of each application.

Thus, a fine grain policy was chosen for the SMILE system, which uses one application as the basic unit for replacement. When a new un-cached application is launched and there is inadequate storage space remaining, we will remove one or more of the least expensive applications to make room for the new one.

We first check the frequency of use of all cached applications and retain the applications used most frequently. The size of the applications is also an important factor. If a large application that is used infrequently is removed, then there will be a very large waiting time and network traffic penalty if the application is re-launched.

Mobile application usage analyses indicates that a user generally uses a very small subset of installed applications. Algorithm 1 is formulated based on the latter consideration.

The actual approach used is slightly different due to an additional updating strategy, described in Section 3.5.

In our system, the application replacement module executes this strategy. This module is a system service that collects the execution information for each application, and stores it in a private storage area. When the remaining storage space is not enough for a new application, this service will use Algorithm 1 to

Algorithm 1: *Replacement*(App_a)

```
Input: App_a
Totol_Size = 0;
initializing Application_Penalty;
initializing Cached_Applications;
for i = 1 to App_a.len do
    if App_a[i].id not in Cached_Applications then
        Application_Penalty[App_a[i].id] =
        Application_Penalty[App_a[i].id] + App_a[i].size;
        Total_Size = Total_Size + App_a[i].size;
        while Total_Size > MAX_STORAGE_SIZE do
            Pick p from Application_Penalty where Application_Penalty[p] is
            smallest;
            Cached_Applications.remove(p);
            Uninstall p;
            Total_Size = Total_Size − p.size;
        end
        Cached_Applications.add(App_a[i].id);
    end
end
```

perform the remove operation. Meanwhile, it will back up the user application data to the remote server.

3.5 Application Updating Strategy

Applications are updated efficiently and seamlessly in our system. This section describes the update strategy of our proposed system.

Both cached and un-cached applications can receive updates at any time. An application update is initiated by the server side; the mechanism is described in Section 3.4.

To update an un-cached application, a lazy update is performed the next time the application is launched. The user will be notified about the update on the home screen. One of the following options is chosen when a cached application is updated.

- For frequently used and/or small applications, a quick update is performed if the change is minor. For a major change, a notification is issued instead of a direct update in case there are compatibility problems with the application data.

- For less frequently used and/or large applications, the application is marked as updated. The policy of replacement was described in Section 3.4. An additional application replacement rule is that applications marked as updated will be considered for removal first.

4 Server

In this section, we discuss server side challenges, including user identification, application subscription and update, storage management, performance, as well as consistency and integrity.

4.1 User Identification

In our system, each user has a logical user identifier *uid*. In mobile computing, every mobile device has a unique IMEI number. To be able to connect to telecommunication networks and for identification, a mobile device needs to have a SIM card, with a unique IMSI number.

When a device runs our proposed system for the first time, a new *uid* is generated, which is associated with both the IMEI and IMSI numbers. These two numbers will subsequently be used for identification, to ensure that the same device and subscriber are using the system. When either of the two numbers changes, the user has to register the new numbers in the system to use their original account.

4.2 Application Subscription and Update

As described in Section 3.1, users need to subscribe to applications in SMILE in order to use them. On the server side, the application subscription information is stored in the MDS. The MDS then generates a subscribed application list for the user, and checks whether the user is authorized to use the applications when the client requests applications and user data.

On server side, when an application is updated by the administrator through the MDS, notifications of application update will be pushed to clients. Then clients will take action according to the policies described in Section 3.5.

4.3 Storage Management

As described in Subsection 2.2, SMILE identifies the user data of an application for a specific user by the (*uid*, *aid*) pair, and stores this piece of user data as a large binary object. Objects are stored among the UDSes in key-value form.

In order to save on storage space, SMILE computes message digests for all the objects stored in the UDSes. We introduce an indirection here: The keys are first mapped to digest values, and then digest values to objects. This process could find identical objects, and remove redundancy in storage space.

4.4 Performance

To optimize performance, the Front End provides a caching mechanism for applications and user data. When FE receives a request for application or user data, it first checks the cache, and returns the data immediately if there is a hit in the cache. In case of a cache miss, FE queries the MDS for the site of the requested data, and then requests the required data from the target AR or UDS.

Multiple FEs can be installed in the system to reduce the workload per FE. Cached data on FEs should be treated carefully here, as application or user data updates may introduce inconsistency. Cached data should be invalidated when it

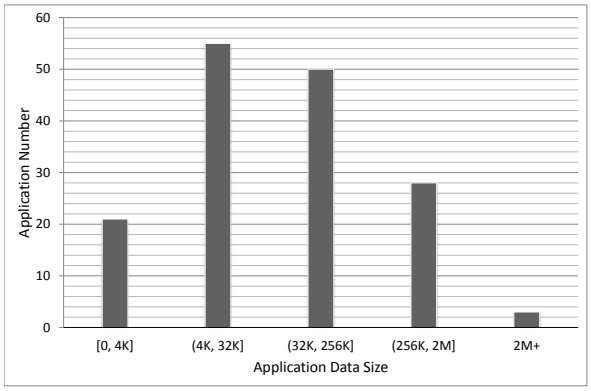

Figure 4 Distribution of size of user data for daily-used applications.

is modified by application upgrades or user data updates. Copies of the data on other FEs should be invalidated as well. We use the write-through policy because the modification of one specific piece of data is rare in practical use.

4.5 Consistency and Integrity

In the context of mobile computing, the data connection can sometimes be slow or unavailable. As mentioned in Section 3, the user data of one application needs to be synchronized on the server side when the application has been replaced. Statistics from a study of several smartphones which were in daily use that analyzed over 150 applications show that user data are typically small (as shown in Figure 4). About 50% of the applications have user data size up to 32kB, and 80% are below 256kB. SMILE transfers data in packets with 32kB in size. This size has been chosen keeping in mind the trade-off between the time overhead due to control packets, and the cost of re-transmission due to communication failure. If the transfer does not go through, the remaining packets will be retransmitted once the data connection is stable again.

According to the data synchronization policy described in Subsection 3.4, user data only needs to be synchronized after the corresponding application has been replaced. Thus, user data is not modified during synchronization, which ensures its consistency.

5 Implementation

The SMILE system can be implemented in several mobile operating systems. For this study, we implemented and evaluated the proposed system on the Android Gingerbread platform with a Wi-Fi network. We used Nexus S smartphones for the experiments. Our implementation is above framework level; therefore, many other Android devices can be supported by our system.

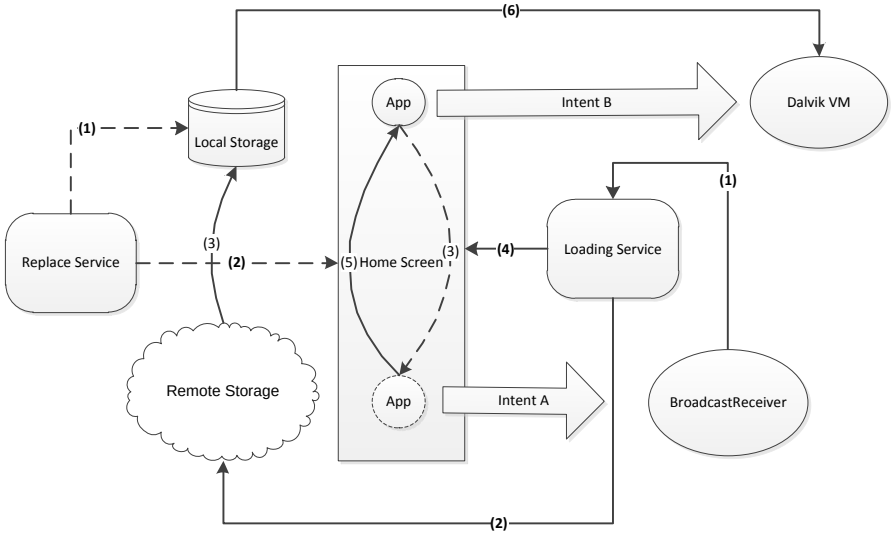

Figure 5 Control flow of On-demand Execution and Replacement.

The Instant Application Subscription is implemented as an application in the Android OS, installed like an App from the Android Market. In this application, users can complete a simple registration to get unique account numbers based on their devices' IMEIs. They can then see the list of all applications and their subscribed applications. Subscribing to an application will result in a quick download of metadata saved to the data directory of TransparentPackageManager. The user can unsubscribe from an application by uninstalling it.

Uniform Application Presentation: In a normal Android system, all application information is collected by the PackageManager, which is an important system manager component inside the runtime framework. We considered changing the implementation for PackageManager to make it retrieve transparent application information through the network. However, this component is very critical for the framework, and any bugs or errors will make the whole system unstable.

Thus, this layer was implemented in the home screen of Launcher, an Android application that creates a new home screen in Android. Launcher directly renders the application labels and icons. We injected our code in the Launcher source code to make it display applications from two different sources. The first source was the original PackageManager, through which cached application information is provided, which the unmodified Launcher supports. The other was our Transparent-PackageManager, a fake package manager that provides the un-cached application information, using only the small metadata retrieved from the server after the first download of the application lists.

Intent is an important component in Android as every icon in the home screen is bound to Intent in memory. The Intent contains the package name and the entry point for an application, which will be used by Dalvik VM to load the application's contents. Under our On-Demand Application Execution paradigm,

there are different Intents for cached and un-cached applications. The Intents for cached applications are unchanged from the default. Intents for un-cached applications contain basic application data and the remote address of the execution code to be fetched. When an un-cached application is launched, Intent A will be sent and a BroadcastReceiver will handle the intent to start a loading Service, which will retrieve the rest of the application data. After the retrieve process, the loading Service will install the application data, create an Intent B containing only the package name and entry point, and then launch it. Thus, the un-cached application is launched.

When an un-cached application is launched, it becomes a cached application. To avoid repeating the loading process during subsequent executions, we replace the Intent bound to the icons in the home screen with Intent B, just like other cached applications.

The implementation of the replacement service is the reverse of the process of transparent application execution. First, the replacement service decides which application(s) are to be deleted from the cache, which are then uninstalled. Finally, the Intent bound to this application is replaced with an Intent of the type that references an un-cached application.

The implementation of the application update process is based on push notifications. The update message is pushed to the client. As described in Section 3.5, the quick update is a reinstall, and the lazy update is just like a normal replacement in the SMILE system.

6 Evaluation

The SMILE system was evaluated through experiments performed in a test environment. PC servers, Android Phones, and WLANs were used to demonstrate the Transparent Application Streaming system.

Samsung Nexus S smartphones were used as clients running Android OS version 2.3. The ROM size for the Nexus S is 512 MB, and the RAM size is also 512MB. The PC servers had Intel i5 processors and 4GB DDR3 RAM. The PC servers were equipped with 7200 rpm Western Digital hard disks.

The following aspects were evaluated:

- Application Loading Performance: whether the execution of an application is transparent to the user, so they do not know if the application is cached locally or stored remotely. We evaluate the loading performance to check if the system is usable.

- Application Replacement Penalty: The case where users have subscribed to more applications than they can store locally. Launching applications will require additional loading time for network communication. We measure the average penalty under *overload* to evaluate the usability of the SMILE system.

We used the 461 top free Android applications from the Android Market(Google 2011*a*) as test applications, and constructed a histogram based on size. Figure 6 shows that most applications are smaller than 16 MB. The average size of the 461 applications was calculated to be 3.55MB.

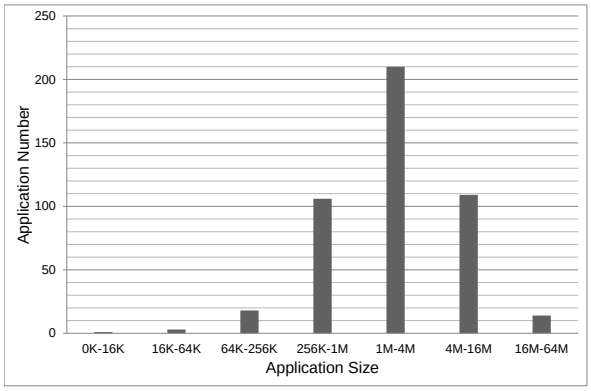

Figure 6 A Histogram of the application size.

6.1 Application Fist Time Loading Performance

As loading speed is very important in terms of the user experience, we conducted experiments to test the performance of loading applications for the first time in our system. We set up the application server and the test clients in a local network, and tested the applications' first loading time on the Nexus S phones running Android version 2.3.7 connected through Wi-Fi. The results are shown in Figure 7.

In Figure 7, every dot represents an application. The X axis is the application size and the Y axis is the application loading time. The following observations can be made:

- The loading time for applications increases linearly with size.
- Most applications take under 10 seconds to load. This waiting time is considered reasonable for the users. Certainly, this test is under very ideal condition, the real situation might be slower. But since the mobile network technology is developing very fast, the future wireless network bandwidth must be much better.

6.2 Application Replacement Penalty

To test the replacement penalty, we first define the usage model. As our system is not deployed widely, we had to test performance through simulation experiments.

First, we define the penalty. We evaluate the penalty under different overload scenarios. Suppose a user subscribes to a set of applications, and their total size is less than the storage limit. In this case, there will be no penalty except when the application is first launched. When the total size of all applications exceeds the storage limit, there will most probably be a cache miss in the local cached list.

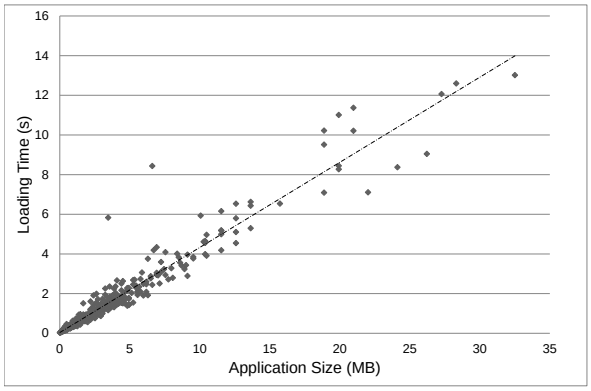

Figure 7 Application Loading Time.

This will impose a penalty of extra loading time and network traffic. We define the penalty as following figures.

$$P_r = \frac{\sum_{i=0}^{n} T(A_i)}{n}$$

Here P_r represents the penalty, which is the extra average loading time spent on each application load operation for an overload rate equal to r. The overload rate is the ratio between the total size of all subscribed applications and the local storage space available. A_i represents the application picked for the ith iteration of the simulation. $T(A_i)$ represents the loading time of application A_i.

Since Figure 7 indicates a linear relationship between the loading time and application size, a secondary penalty equation does not need to be defined. The final equation is given by:

$$P_r = \frac{\sum_{i=0}^{n} S(A_i^-)}{n * D_{avg}}$$

Here A_i^- is the application that is not cached in the ith iteration of the simulation. $S(A_i^-)$ is the size of the Application A_i and D_{avg} is the average download speed.

As our experiments are simulations, we do not assume an average download speed. Instead, we use the cache miss rate to show the effect of a replacement, which is defined as:

$$C_r = \frac{\sum_{i=0}^{n} S(A_i^-)}{\sum_{i=0}^{n} S(A_i)}$$

where, C_r is the cache miss rate under overload ratio r. The denominator denotes the total of all application sizes in the loading sequence. Once the sequence is generated, $\sum_{i=0}^{n} S(A_i)$, n and D_{avg} are fixed, which implies that C_r and P_r have a linear relationship.

Next, we have to define the application characteristics. As we do not have access to the real usage data, we make our own hypothesis. We generate the application size distribution to fit the distribution in the Figure 6, where the average size is 3.55MB.

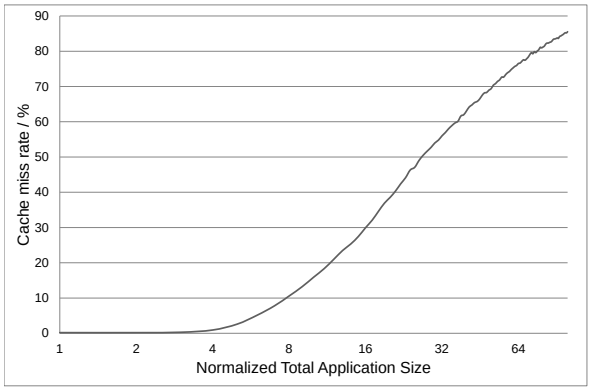

Figure 8 Cache miss rate under overload conditions.

To find the usage frequency of each application, we use the research results from (Verkasalo 2009). The data in abstract of this paper tells us that the top 5% of applications typically represent over 90% of the total application usage, and the bottom 80% of applications represent just 2.10% of the total usage. We chose an exponential distribution function that is close to the observed results presented in this paper:

$$f(x) = 20e^{-20x}$$

Under this distribution, the top 5% applications will represent 60% of the total application usage while the bottom 80% applications represent 20% of the application usage, which is a little more conservative than the distribution discussed in (Verkasalo 2009).

In our test bed, thousands of dummy applications of various sizes and frequency attributes were generated. We then generate the sequence of launching applications by using Round-robin scheduling, applications with higher frequency will be probably used more frequently. We then simulated the sequence involved in launching applications as given in Algorithm 1. We record the cache miss penalty for each iteration of application execution, under specific overload situations.

The curve in Figure 8 represents the relationship between the overload percentage and replacement penalty. The X-axis represents the normalized total application size that ranges from 1 to 100. This means that the total size of all subscribed applications is 1 to 100 times larger than the local storage space. The Y-axis represents the cache miss rate.

The results show that for an overload ratio below 10, the cache miss rate is around 10%. This represents a user with a 16GB Android device using the SMILE system for application storage, who has subscribed too many applications, whose total size is around 160GB. The user will incur an average extra loading penalty of 0.35MB if their usage pattern is as described in (Verkasalo 2009), based on our experimental results. If we assume that the average download speed is the same as the result in Section 6.1, them it means it will averagely cause an extra load time around 0.1 to 0.2 seconds when we launch an application.

7 Related Work

Application streaming is one of several approaches proposed for application management improvement in mobile devices. This section compares SMILE with existing approaches, and evaluates it in the context of mobile/pervasive computing, especially automatic partitioning, and remote execution.

Transparent computing stores the entire user environment externally, including the operating system, user applications, and, user data, and the thin client translates each disk operation into a network event (Zhang et al. 2010, Tian et al. 2009, Zhang and Zhou 2006, 2009). Although SMILE is mainly designed based on transparent computing principles, the application cache policies are its distinguishing feature. As transparent computing streams every bit over the network, it involves heavy network traffic, and hence, is more suitable for wired networks (e.g. LAN) (Liu et al. 2010). SMILE is targeted at mobiles and wireless networks, and therefore, network latency, bandwidth, and power consumption are important factors. SMILE caches the most frequently used applications to improve user experience even when there is no network connection.

The Apple App Store (Apple 2011*a*) and the Android Market (Google 2011*a*) track which applications are installed by users, but they do not install or backup applications automatically. Thus, users need to have the knowledge and skills needed to synchronize applications and data manually. SMILE greatly simplifies this process and reduces the effort required. It also guarantees that user applications and data will always be available when the device is connected to a network.

Furthermore, there has been substantial prior research on the topics of partitioning and remote execution. The CloudClone (Chun et al. 2011, Chun and Maniatis 2010) partition applications uses a framework that combines static program analysis with dynamic program profiling, and executes remote analysis using a virtual machine. The MAUI (Cuervo et al. 2010) requires programmers to annotate methods as `REMOTABLE` to tell the runtime process to perform remote code execution. SMILE complements such systems. CloudClone and MAUI assume that all applications are already present in the local flash memory, whereas SMILE abstracts them to make it appear that they are.

Remote execution addresses the problem of executing resource-intensive applications on resource-poor hardware. These applications are pre-partitioned between local and remote execution (Balan et al. 2007, 2002)(Su and Flinn 2005, Flinn et al. 2003, Balan et al. 2002, Flinn et al. 2001). ISR (Satyanarayanan et al. 2005) leverages the checkpointing and restart ability of the virtual machine to allow an application to be suspended on one machine and resume on another machine. As mentioned earlier, SMILE considers the application as a whole and does not conflict with the remote execution process.

Dropbox (Dropbox 2011), iCloud (Apple 2011*b*), and Google Sync (Google 2011*b*) are three representative services providing user data management in mobile. Dropbox provides a cloud storage service that lets users store and share files across the Internet using file synchronization. Dropbox, when installed on desktops and laptops, can synchronize local files and directories, and mirror the content between local and cloud storage. On mobile platforms, Dropbox maintains a cache of the most accessed files, according to the user's preferences. However, Dropbox

is unaware of the application that the data belongs to, while SMILE knows which application owns which files.

iCloud (Apple 2011*b*) aims to provide a service for storing personal music, photos, documents, and more, and wirelessly pushes the content to all devices, automatically, and seamlessly. Programmers have to use dedicated APIs to enable iCloud data synchronization for a particular application. Google sync only synchronizes information inside the Android system, including contacts, emails, calendars, and other content. It also provides a backup service for Android, which allows developers to copy persistent application data to remote cloud storage, to provide a restore point for the application data and settings. If a user performs a factory reset or moves to a new Android-powered device, the system automatically restores the previously backed up data when the application is re-installed. In contrast to iCloud and Google Sync, SMILE synchronizes data transparently for both users and programmers, and enables the automatic reinstallation of applications.

8 Conclusion and Future Work

In this paper, we proposed a new paradigm for mobile cloud computing. The SMILE system lets users use all subscribed applications transparently, as if they are all installed locally. Applications are also updated efficiently and promptly, with guaranteed data synchronization. A prototype system was developed and evaluated to demonstrate the feasibility of our proposed system. The experimental results demonstrate that our system is very effective in handling significant amounts of overload in terms of installed applications. Thus, SMILE is a feasible system for future mobile cloud computing infrastructure.

However, our system has some limitations. One limitation is that all application metadata is stored locally (the icon, label etc.). When the number of applications grows significantly, the size of metadata will become significant and lead to wasted storage space.

Our future work may include improvements to metadata storage, cache granularity and replacement policy, improved power management and support for other operating systems (such as iOS). Finally, further research is needed to provide security and data protection for the communication and cloud storage systems.

Acknowledgement

We thank the anonymous reviewers for their useful feedback. The research was partially supported by National Key Science and Technology Initiative Grant No. 2009ZX01039-001-001, National Natural Science Foundation of China Grant 61073175, 61035004, and National 863 High Tech Plan Grant 2012aa012601.

References

Apple (2011*a*), 'App Store', `http://www.apple.com/iphone/from-the-app-store/`.

Apple (2011*b*), 'iCloud', `https://www.icloud.com/`.

Balan, R. K., Flinn, J., Satyanarayanan, M., Sinnamohideen, S. and Yang, H.-I. (2002), The case for cyber foraging, *in* G. Muller and E. Jul, eds, 'Proceedings of the 10th ACM SIGOPS European Workshop, Saint-Emilion, France, July 1, 2002', ACM, pp. 87–92.

Balan, R. K., Gergle, D., Satyanarayanan, M. and Herbsleb, J. D. (2007), Simplifying cyber foraging for mobile devices, *in* E. W. Knightly, G. Borriello and R. Cáceres, eds, 'Proceedings of the 5th International Conference on Mobile Systems, Applications, and Services (MobiSys 2007), San Juan, Puerto Rico, June 11-13, 2007', ACM, pp. 272–285.

Chun, B.-G., Ihm, S., Maniatis, P., Naik, M. and Patti, A. (2011), Clonecloud: Elastic execution between mobile device and cloud, *in* 'European Conference on Computer Systems, Proceedings of the Sixth European conference on Computer systems, EuroSys 2011, alzburg, Austria - April 10-13, 2011'.

Chun, B.-G. and Maniatis, P. (2010), Dynamically partitioning applications between weak devices and clouds, *in* '1st ACM Workshop on Mobile Cloud Computing and Services (MCS 2010)'.

Cuervo, E., Balasubramanian, A., ki Cho, D., Wolman, A., Saroiu, S., Chandra, R. and Bahl, P. (2010), MAUI: Making smartphones last longer with code offload, *in* S. Banerjee, S. Keshav and A. Wolman, eds, 'Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services (MobiSys 2010), San Francisco, California, USA, June 15-18, 2010', ACM, pp. 49–62.

Dropbox (2011), 'Simplify your life', `http://www.dropbox.com`.

Flinn, J., Narayanan, D. and Satyanarayanan, M. (2001), Self-tuned remote execution for pervasive computing, in *Proceedings of HotOS-VIII: 8th Workshop on Hot Topics in Operating Systems, May 20-23, 2001, Elmau/Oberbayern, Germany Proceedings of HotOS-VIII: 8th Workshop on Hot Topics in Operating Systems, May 20-23, 2001, Elmau/Oberbayern, Germany* (2001), pp. 61–66.

Flinn, J., Sinnamohideen, S., Tolia, N. and Satyanarayanan, M. (2003), Data staging on untrusted surrogates, in *FAST Proceedings of the FAST '03 Conference on File and Storage Technologies, March 31 - April 2, 2003, Cathedral Hill Hotel, San Francisco, California, USA* (2003).

Google (2011*a*), 'Android Market', `https://market.android.com/`.

Google (2011*b*), 'Google Sync', `http://www.google.com/mobile/sync/`.

Ligang, X. (2010), '78% of iphone users never install applications', `http://xiangligang.blog.techweb.com.cn/archives/210.html?1321668111`.

Liu, H., Zhang, Y., Zhou, Y. and Xue, R. (2010), A rate and resource detection based receive buffer adaptation approach for high-speed data transportation, in *Proceedings of the 19th International Conference on Computer Communications and Networks, IEEE ICCCN 2010 Proceedings of the 19th International*

Conference on Computer Communications and Networks, IEEE ICCCN 2010, Zürich, Switzerland, August 2-5, 2010 (2010), pp. 1–6.

Poulsen, T. (2011), 'Installing Android apps to the SD card', `http://developer.appcelerator.com/blog/2011/07/installing-android-apps-to-the-sd-card.html`.

Proceedings of HotOS-VIII: 8th Workshop on Hot Topics in Operating Systems, May 20-23, 2001, Elmau/Oberbayern, Germany (2001), IEEE Computer Society.

Proceedings of the 19th International Conference on Computer Communications and Networks, IEEE ICCCN 2010, Zürich, Switzerland, August 2-5, 2010 (2010), IEEE.

Proceedings of the FAST '03 Conference on File and Storage Technologies, March 31 - April 2, 2003, Cathedral Hill Hotel, San Francisco, California, USA (2003), USENIX.

Purdy, K. (2010), 'What Should I Do When My Android Runs Out Of App Space?', `http://www.lifehacker.com.au/2010/10/what-should-i-do-when-my-android-runs-out-of-app-space/`.

Satyanarayanan, M., Kozuch, M., Helfrich, C. and O'Hallaron, D. R. (2005), 'Towards seamless mobility on pervasive hardware', *Pervasive and Mobile Computing* **1**(2), 157–189.

Su, Y.-Y. and Flinn, J. (2005), Slingshot: deploying stateful services in wireless hotspots, *in* K. G. Shin, D. Kotz and B. D. Noble, eds, 'Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services (MobiSys 2005), June 6-8, 2005, Seattle, Washington, USA', ACM, pp. 79–92.

Tian, P., Zhang, Y., Zhou, Y.-Z., Yang, L. T., Zhong, M., Weng, L. and 0002, L. W. (2009), 'A novel service evolution approach for active services in ubiquitous computing', *Int. J. Communication Systems* **22**(9), 1123–1151.

Verkasalo, H. (2009), Open mobile platforms: Modeling the long-tail of application usage, *in* 'Fourth International Conference on Internet and Web Applications and Services', pp. 112–118.

Zhang, Y., Yang, L. T., Zhou, Y. and Kuang, W. (2010), 'Information security underlying transparent computing: Impacts, visions and challenges', *Web Intelligence and Agent Systems* **8**(2), 203–217.

Zhang, Y. and Zhou, Y. (2006), Transparent computing: A new paradigm for pervasive computing, *in* J. Ma, H. Jin, L. T. Yang and J. J. P. Tsai, eds, 'Ubiquitous Intelligence and Computing, Third International Conference, UIC 2006', Vol. 4159 of *Lecture Notes in Computer Science*, Springer, pp. 1–11.

Zhang, Y. and Zhou, Y. (2009), *Transparent Computing: Concepts, Architecture, and Implementation*, Singapore: Cengage Learning, 2009.