swt 512_4

by Vinod kumar

Submission date: 17-Feb-2020 11:31AM (UTC+0530)

Submission ID: 1258761433

File name: Thesis_SWT_512.v4.docx (817.7K)

Word count: 4030

Character count: 24024

A

Dissertation On

Storage Optimization using resource pooling

Submitted in Partial Fulfillment of the requirement

For the Award of Degree of

Master of Technology

In

Software Technology

Submitted By

Pratush Kumar Srivastava
University Roll No. 2K16/SWT/512

Under the Esteemed Guidance of

Dr. Vinod Kumar
Associate Professor, Department of Computer Science & Engineering, DTU, Delhi

2017-2020(Jan)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY
DELHI – 110042, INDIA

Delhi Technological University
(Government of Delhi NCR)
Bawana Road, Delhi- 110042

CERTIFICATE

This is to certify that the dissertation titled **"Storage Optimization using resource pooling"** is a bonafide record of work done at **Delhi Technological University** by **Pratush Kumar Srivastava, Roll No. 2K16/SWT/512** for partial fulfillment of the requirements for the degree of **Master of Technology** Degree in **Software Technology** in the **Department of Computer Science & Engineering**. This project was carried out under my supervision and has not been submitted elsewhere, wither in part or full, for the award of any other degree or diploma to the best of my knowledge and belief.

Dr. Vinod Kumar
Associate Professor,
Department of Computer Science & Engineering, DTU

Date: __________

ACKNOWLEDGEMENT

I would like to express sincere thanks and respect towards my guide **Dr. Vinod Kumar, Associate Professor, Department of Computer Science & Engineering, Delhi Technological University Delhi**, who have supported and encourage me during the course of the project. This work could not have been accomplished without your guidance.

I consider myself very fortunate to get the opportunity to work with him and for the guidance and support I have received from him. The completion of the project is possible with his regular guidance. Special thanks for not only providing me necessary project information but also teaching the proper way and techniques of representation.

I am grateful to DTU for providing good support and right resource and environment for this work to be carried out.

Pratush Kumar Srivastava
University Roll No: 2K16/SWT/512
M.Tech (Software Technology)
Department of Computer Engineering
Delhi Technological University
Delhi-110042

ABSTRACT

Now days there are a plethora of mobile applications available in the market like Google Playstore, and Galaxy Store. A huge number of applications are being added to this list each day. As the demand for user engagement increases, more and more rich graphical contents are provided by applications. In order to have a global footprint, applications provide support for multiple languages. All this considerably increases the size of the applications.

Users install several applications on their mobile devices which results in shortage of memory space in the devices. Each application uses similar text or resources at application layer for display interface.

We have proposed a technique to increase the usable memory space in the devices by leveraging the fact that each user does not require multiple languages simultaneously and many applications have duplicate/common resources.

This thesis proposes a common resource pool for all installed applications to optimize their memory requirements in user's device.

TABLE OF CONTENTS

Chapter 1: Introduction

1.1 PROBLEM STATEMENT

Smartphone is necessity of today's life for all age. Its usage is now in almost every field. Many applications are available based on user interests or need. Due to much application installation in device, there is always the memory limitation to carry more applications. Also due to high resolution of resources and to launch the application globally companies are adding most of local/global language resources which is causing increase apk size. Installing larger number of applications device memory increase which degrade the mobile performance. In this thesis we are providing a memory optimization technique which is based on shared resources for the applications. We call it as the **R**esource **P**ooling **F**ramework (RPF).

1.2 WHAT ARE THE STEPS INVOLVED IN RESOURCE POOLING

Android architecture are having various resource components like String Resource, Layout Resource, audio, animations etc.

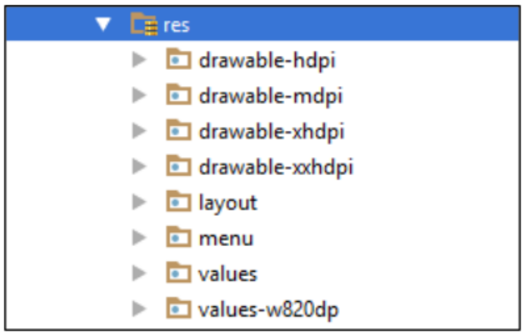


Fig. 1.1: Android Resource Package

These resources are used be one application and are present in the bundle called application package or .apk.

In the first step we need to create a centralized resource pool for each type of resource at framework level.

This will store different types of resources.

In the next step, while installing any new application, separate the code file and resource file.

In the thirst step, parse the resource file to identify different types of resources.

In the fourth step we compare each component of the parsed resource file with the resource pool.

If we find the duplicate match, we create a new ID associated with the application which is just mapped to the resource pool.

If we do not fid the match, we add this resource in the resource pool and create a new ID. Map it with the application and resource pool.

Now when an application request for any resource, it always pick the ID form resource pool.

1.3 THESIS MOTIVATION AND GOAL

With the increased usage of smartphone and availability of too many necessary applications, user install most of the application and most of the application are actually sometime used app, but they installed in the device and part of device memory. Due to the heavy graphic and better look and feel app developers uses high resolution images and to increase the download, app developer always try to add as much as possible local language support.

Due to many apps installed in the devices, people are started feeling the memory crunch in his device. As we can see many apps are using same/similar resources of common items or language, like back icon, undo icons, app specific icons (identity icons) and common languages.

To provide better space to user, common resources should be managed in the way to remove duplication of content and utilize maximum space as much as possible.

This thesis is based on a new approach for resource sharing among applications this will help to avoid duplicity of the common resource image and string present in different language.

Which results in memory saving.

So it always makes sense to develop an architecture which saves the memory and give better user experience with more number of applications for the user.

1.4 THESIS DETAILS IN DEPTH

This thesis is divided into six different parts.

Chapter 1 defines the problem statement for the thesis, which is related with the device performance impact and memory problem due to too many apps installation from Google play store, too many apps required too much memory in device. The solution is to provide as much as free space to user by removing duplicate resources from application.

Chapter 2 describes the related work done in the areas of memory optimization in device and techniques available in the market. This thesis deals in optimization of storage through common resource pool maintained by system/framework during application installation.

Chapter 3 explains the research topics used in detail. In our research details, we choose some native apps to perform our algorithm and test our solution result. Also we defied the area and resources on which we will apply our algorithm to test the improvement.

Chapter 4 is explaining the proposed approach for the solution. This section of the thesis covers the various steps involved to develop the Resource pool framework architecture.
We explain the details by when and how the RPF will start the pooling of resources in to single repository and methods of installation of application and resource modification during apk installation process.

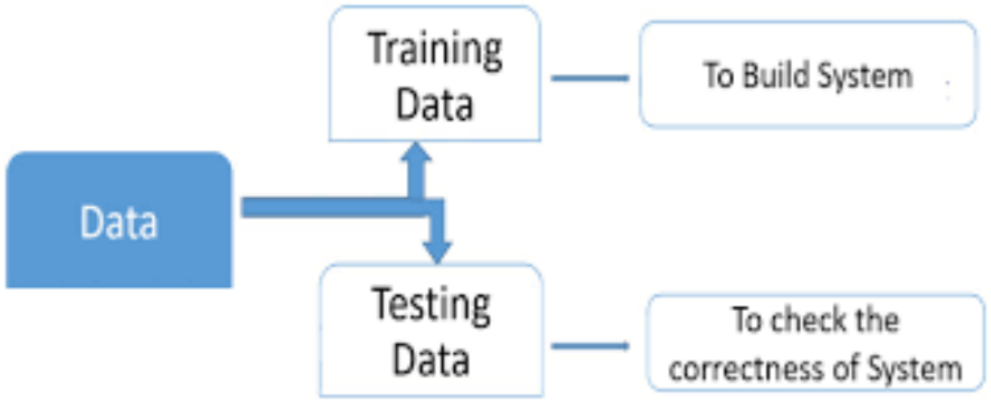
Chapter 5 illustrates the step by step results of before and after comparison of memory and database size. As the test was performed with limited set of native application, we checked the side of apk before and after applying our solution. We also included the total memory size of the device before and after the solutions so that we can include the Resource pool size in the final calculation.

Chapter 6 is the conclusion of the thesis. It describes the benefits and improvement of the single resource pool, which helps to remove duplicate/unnecessary resources from the system and user will get more space to download many apps and use the extra space provided by the RPF.

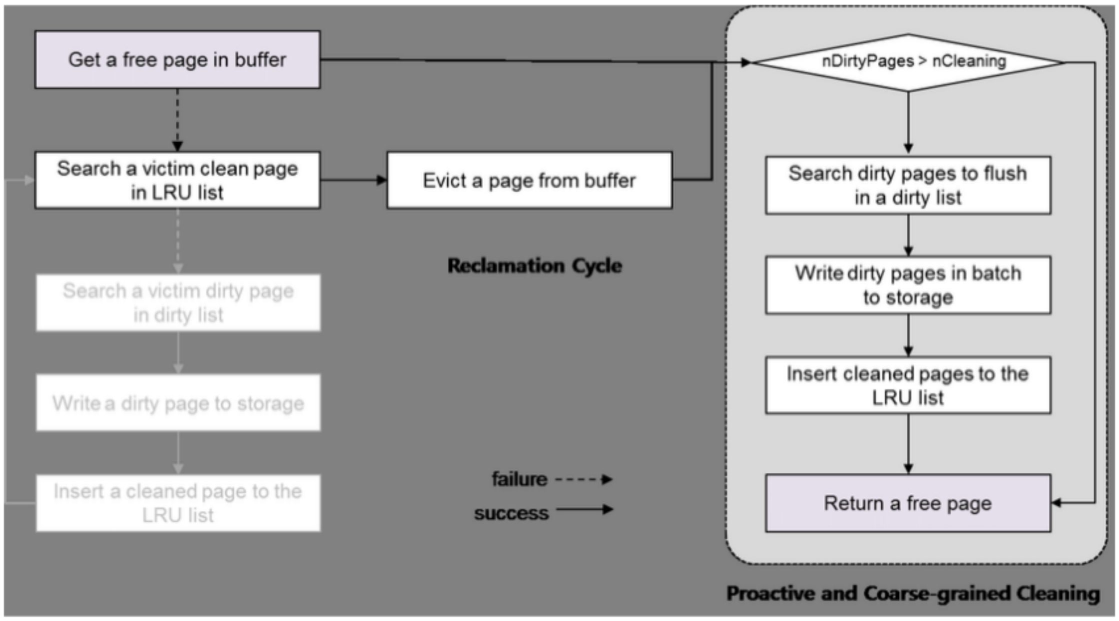
Chapter 2: Related Work

2.1 APPLICATION REMOVAL BASED ON USER USES PATTERN:

One of the memory management paper[17] discusses about user application usage patterns and then analyze this data to decide whether application should be removed from device memory or not and based on the result through which the process defined background process and update the cached limit dynamically according to application interest of user. This behavior shows significant improvement over memory usages, although, does not completely solve this problem. There are few cases where this technique will fail.



2.2 COMPARES LRU BASED RECLAMATION SCHEME:


Get a free page in buffer
Search a victim clean page in LRU list
Evict a page from buffer
Search a victim dirty page in dirty list
Write a dirty page to storage
Insert a cleaned page to the LRU list
Reclamation Cycle
failure
success
nDirtyPages > nCleaning
Search dirty pages to flush in a dirty list
Write dirty pages in batch to storage
Insert cleaned pages to the LRU list
Return a free page
Proactive and Coarse-grained Cleaning

[18] Compares LRU based reclamation scheme is the process which is proposed intelligent solution through which proposed memory reclamation service that is based on Markov Decision Process (MDP) and simulation which demonstrated the result which shows that the proposed technique surely can effectively learn the user patent or user behavior & loading latency improvement over 22.1% from the LRU-based reclamation scheme on average.

2.3. THREAD-ORIENTED MEMORY MANAGEMENT LAYER (TOMML):

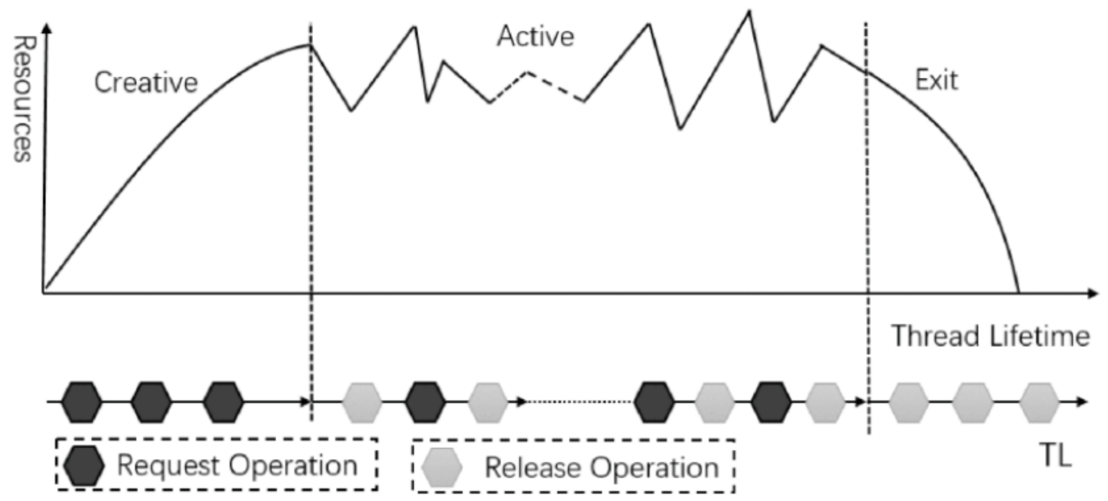


Fig. 2.2: Thread oriented memory management Layer

[19] Describes a memory resource management technique works on OS layer. It works operating system (OS) layer of mobile computing, called the thread-oriented memory management layer (TOMML). According to which if micro kernel architecture pattern will be followed, this can meet user requirement to select plug-ins for achievement of different goals set for optimization. The result shows TOMML improved efficiency of memory allocation by 12%-20%.

2.4 PROPOSES AUTOMATIC REFERENCE COUNTING (ARC)

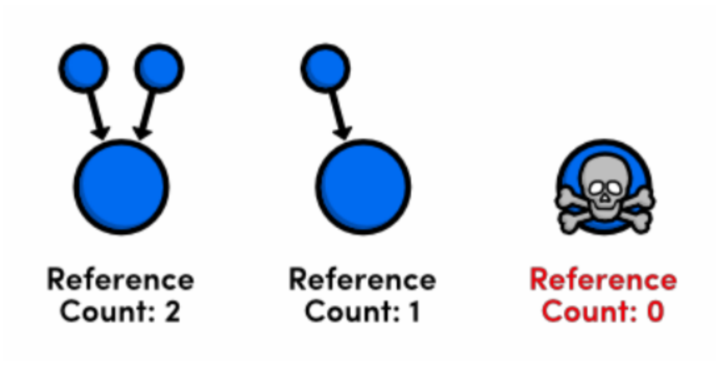
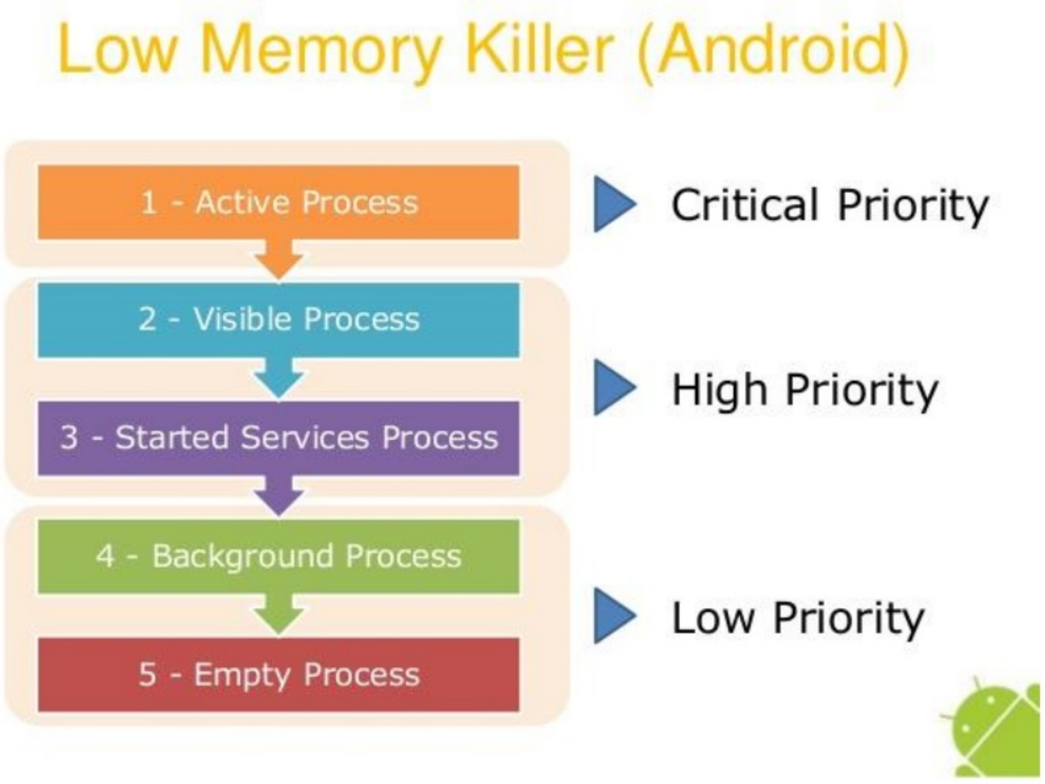


Fig. 2.3: Reference counting approach

[16] Proposes Automatic Reference Counting (ARC) in Android devices instead of Garbage Collector for improvement of memory issues.

2.5 LOW MEMORY KILLER (LMK)



2.5 Low memory killer (LMK)

4

[14] Proposes a new technique to optimize the low memory killer with the help of reinforcement learning. This new low memory killer is an autonomic decision maker for an uncertain environment. It observes various metrics and indicators continuously for better management of

device Memory and takes decisions related to process-killing. It considers launch latencies of apps as penalties in the decision making environment.

Though all of above memory management solution works on Android kernel level & OS or framework layer but none of them work on Application level. This Thesis proposes a solution to work on application level. Next section will describe the technique and algorithm used to solve this problem at application level.

Chapter 3: Research Background

3.1 ANDROID OPERTING SYSTEM:

Android is open operating system which was specifically designed for mobile/smart phone, devices. Android is based on Linux kernel and various open source libraries.

There are various layers in android operation system:

Linux kernel

The first bottom layer of android operating system is based in Linux kernel. This layer provided a connection/bridge between device hardware and device basic essentials like camera, keypad, display etc. Also the major work of Kernel handles the networking device drivers, and other peripheral hardware.

Libraries

All the library components (.so files) are present on the top of the android kernel layer.

These libraries are either provider by the Linux, driver components or open source libraries. Library which are providing the set of apis and communication mechanism between android framework and hardware components. Some more libraries like SQL lite database libraries are provided in the library section to provide the storage medium in devices.

Some major libraries are added which interact with other hardware components like audio, video and camera etc.

Android Runtime

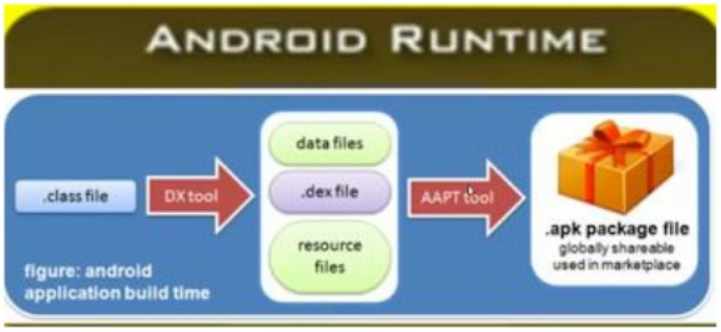


Fig. 3.1: Android Runtime (ART)

The thord layer of android operating system architecture and we can say that the second layer from the bottom. This layer is the key component of android operating system, this major component is called Dalvik Virtual Machine. DVM is similar with Java Virtual Machine (JVM) and it is specially designed for android operating system to optimize the memory (RAM & RMO) to provide the best performance in limited memory devices.

The Dalvik Virtual Machine use Linux core features like memory management and multi-threading, which is based on the Java language which is the key feature of JVM as well. Similar like JVM, DVM is also provide its on process for each android applications as Android apps are running on its own process which is created by Linux zygote and application runs on its own instance of the Dalvik virtual machine.

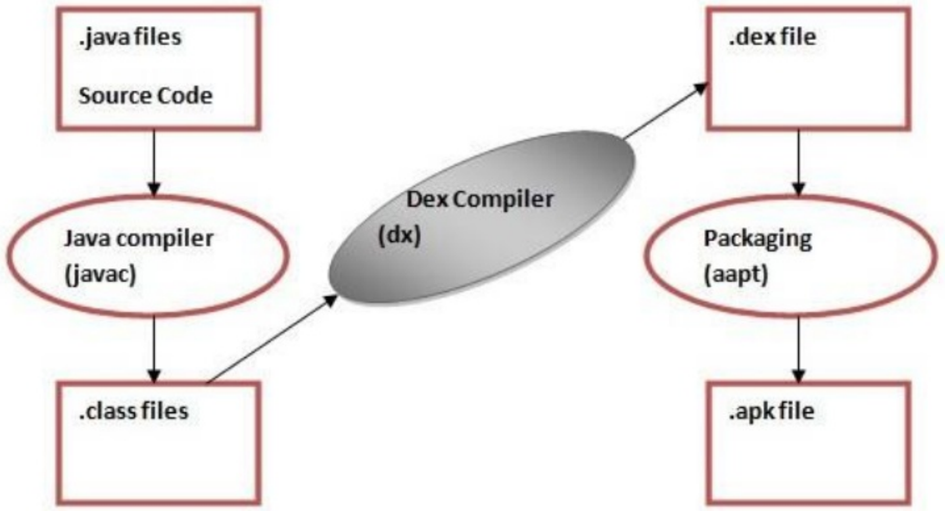


Fig. 3.2: Dalvik virtual machine (DVM)

The Android runtime (ART) is the process which helps to provide the set of core libraries which are enables developers to write the code on java and with the help of Android Framework library Application communicate with android native libraries with Java Native Interfaces (JNI) to send the command to hardware and receive the same at application layer.

Application Framework

The Application Framework is the major component of android which provided the liberary and APIs to provide the support and interface to developer to create a application/User interface and provide the best experience to the end user. The Android framework is mainly depending on the following key services −

- **Activity Manager** – Activity manager is major component of android operating system which controls the screen/UI and manage its life cycle and activity stack.

- **Content Providers** – It is the unique function of android OS, it is allow the application to open its private database/content with limited access.
- **Resource Manager** – It is used to enable sharing of application resources which contains strings, theme, colors and layout(designed by the application developer).
- **Notifications Manager** – Notification is the feature which send the alerts to user when application required to display some attention. To create notification, notification manager system provides the interface to show the same.
- **View System** – View system is major component of Android framework. This will help to provide the predefined set of view/layouts and widget which helps developer to create rich UI for an application.

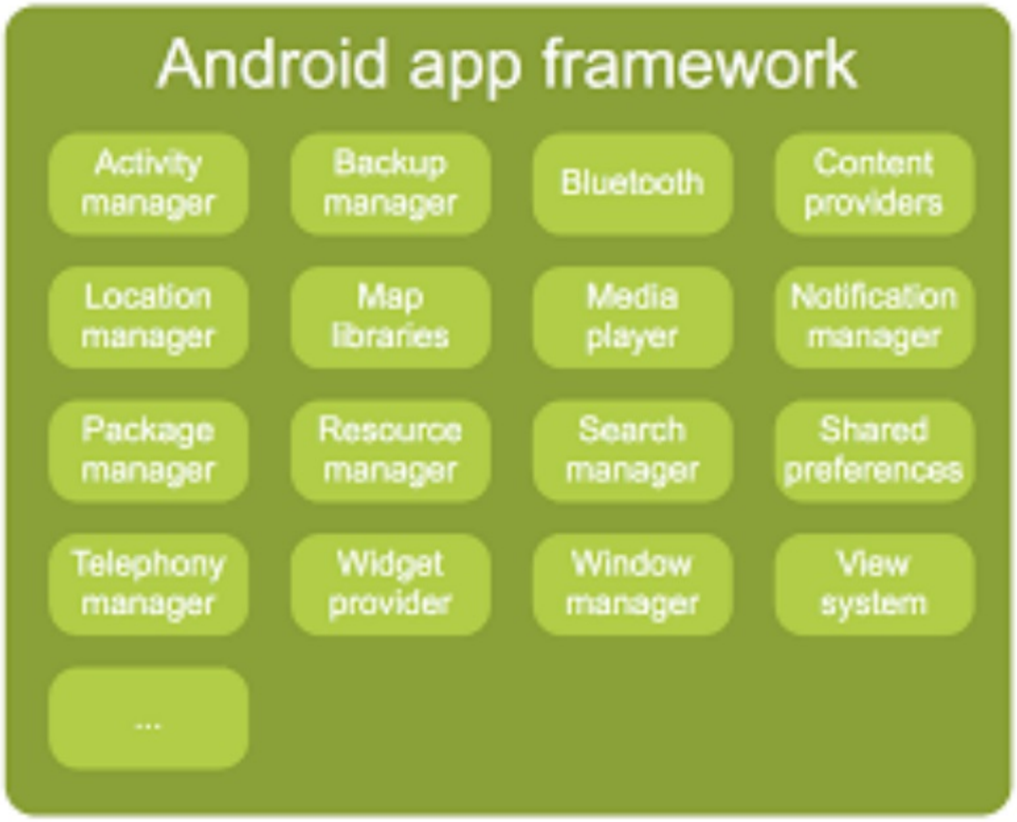
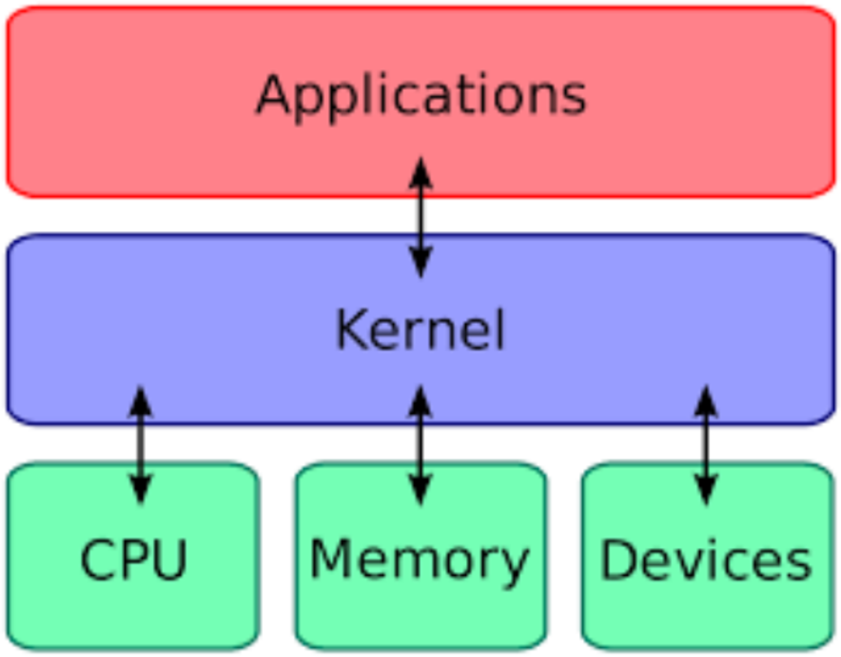


Fig. 3.3: Android framework

Applications

This is the most useful feature provided by the operating system. Application is the program or software which is designed for specific task. Developer allows to writes the code of application in JAVA but the same has been installed only at application layer in device. Few examples of applications are Contacts, Email, Message, Knox , Browser, Games etc. Android partners/ OEMs (original equipment manufacturer) can only access the internal layers including application layer.



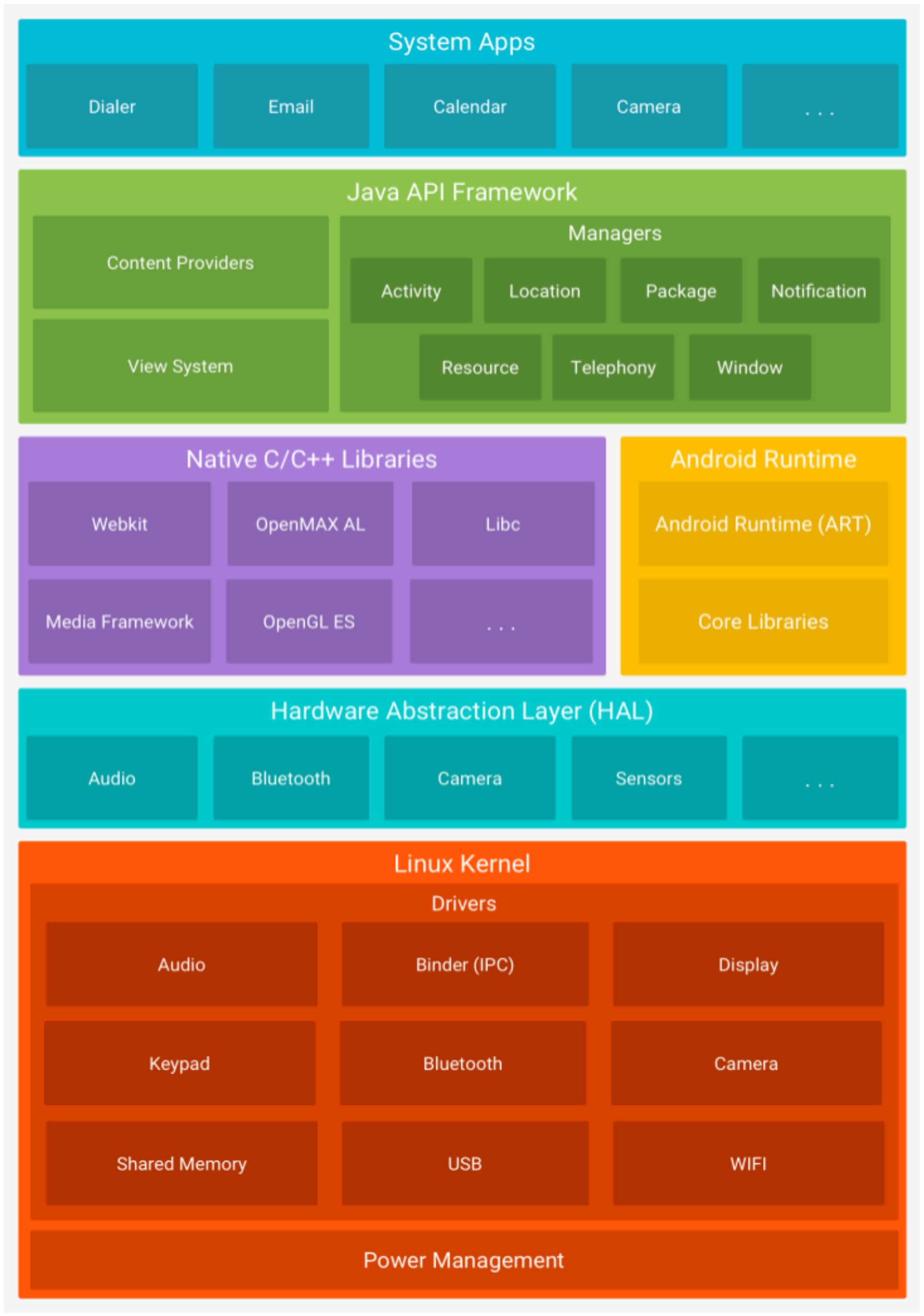


Fig. 3.4: Android Architecture

3.2 CURRENT ARCHITECTURE:

AAPT (Android Asset Packaging Tool)[1] is a build tool that Android Studio [3] and Android Gradle Plugin [7] [2] use to compile and package [5] android app's resources [4]. AAPT parses, indexes, and compiles the resources into a binary format that is optimized for the Android platform. During build time, AAPT collects all of the resources defined by the app and assigns unique resource IDs to them. A resource ID is a 32-bit number in the form of: 0xPPTTEEEE.

PP: represents the package this resource belongs to. (This value is always 0x7f for application resources)

TT: represents the type of the resource (eg. attr, integer, string, drawable, layout, dimen, etc...)

EEEE represents the entry/name of the resource.

The TT and EEEE values are assigned by AAPT arbitrarily. For example, if we have these resource files in an application and it is handled by AAPT in this order.

layout/activity.xml

drawable/android_icon.xml

values/strings.xmlwith

value<stringname="app_name">TestApplication</string>

layout/fragment.xml

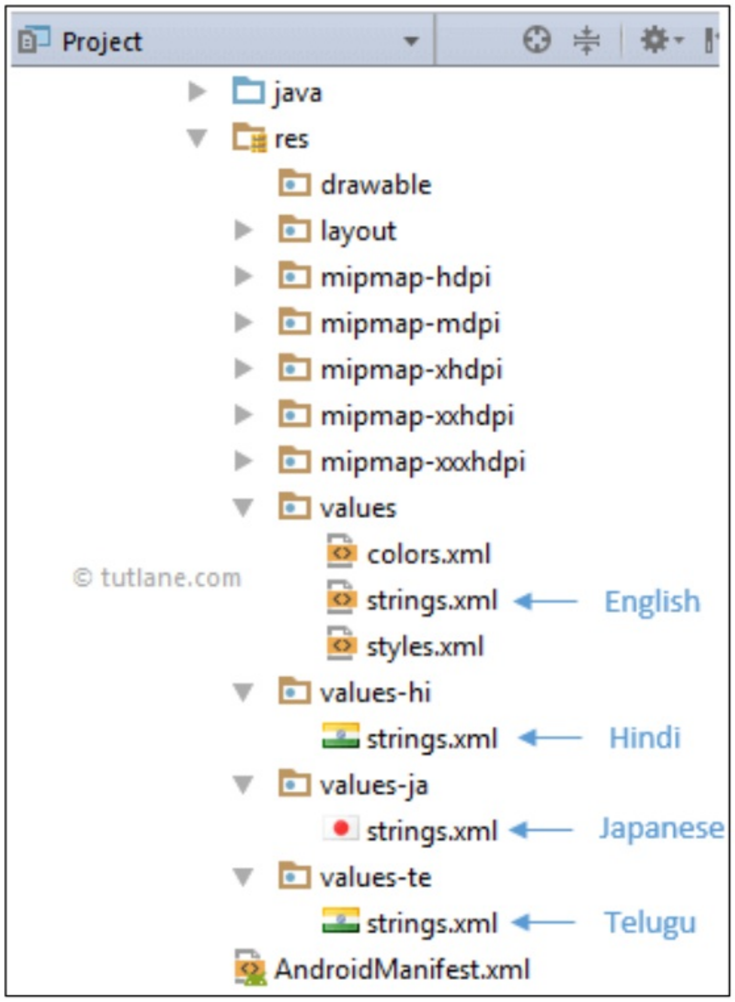


Fig. 3.2: Android Resources with locales strings

3.3 RESOURCE COMPILATION PROCESS:

Step 1: AAPT reads ”layout” so that TT == 01. The first entry/name under that type is ”activity” so that EEEE == 0000. So the final resource ID will be 0x7f010000.

Step 2: Next it reads ”drawable” so that TT == 02. The first entry/name for that type is ”android icon” so that EEEE == 0000. The final resource ID is 0x7f020000.

Step 3: Next it reads ”values/strings.xml” so that TT == 03. The first entry/name for that type is ”app name” so that EEEE == 0000. The final resource ID is 0x7f030000.

Step 4: Next it reads another ”layout” which has already TT == 01. This has a new entry/name ”fragment” so that EEEE gets the next value which is 0001(EEEE == 0001). The final resource ID is 0x7f010001.

3.4 PROPOSED SOLUTION (RPF): We are proposing the solution at framework layer by providing the resource pool which helps to improve the memory management with such a efficient way to reduce the duplicity of resources while installing of application.

As a proof of concept, we experimented by creating a RPF for only string resources. We applied this RPF with only 6 pre-defined applications **(Contacts, Calendar, Memory saver Clock package, Keyboard and Settings).** For this, we modified the Resource and Asset Manager Framework modules to redirect all string related resource requests to our RPF for these applications. Here we managed the pool of string resources.

Chapter 4: Proposed Approach

We propose a method to pool [11] all the application resources at a centralized location called Resource Pool. All the resources will be managed by a Resource Pool Framework (RPF). It will be the responsibility of this RPF to store, retrieve and manage all the resources provided by all the applications installed on the device at any given point of time.

When any of these (test apps) apps is installed, we modified the parsing logic of ”resources.arsc” file from APK to send all string resources to the RPF. Here we checked if the string resource already exists in our pool. If so, we add the package id of the app with that string and insert new mapping between this app res id and res table res id. We used a HashMap for maintaining these mappings with key as app res id and value as res table res id.

When any of the pre-defined applications request a resource using its app res id, the RPF compares its PP value. We only handle the request if value is 7f(for applications). RPF fetches the value from the mapping HashMap with the app res id as key. This value is then checked in the resource pool. First we filter the string resources from the pool using the PP value, then we further narrow it down using the TT value. Lastly, we match the EEEE value and return the requested string resource.

When a new application is installed, the RPF will dynamically assign a package id to this app, parse the ”resources.arsc” file from the APK and update the resource table in the Resource Pool

[Fig. 3]. It will make new entries for only those resources that are not present in the resource pool. For existing resources, it will mark that the resource is being used by this application using the dynamically generated PP value. All new entries will be saved using the translated PPTTEEEE values. RPF will also maintain a mapping between app res id and the resource pool res id. So for resources that are being used by multiple applications, the RPF will have a single entry of translated res id along with the package ids of all the apps that are using this resource.

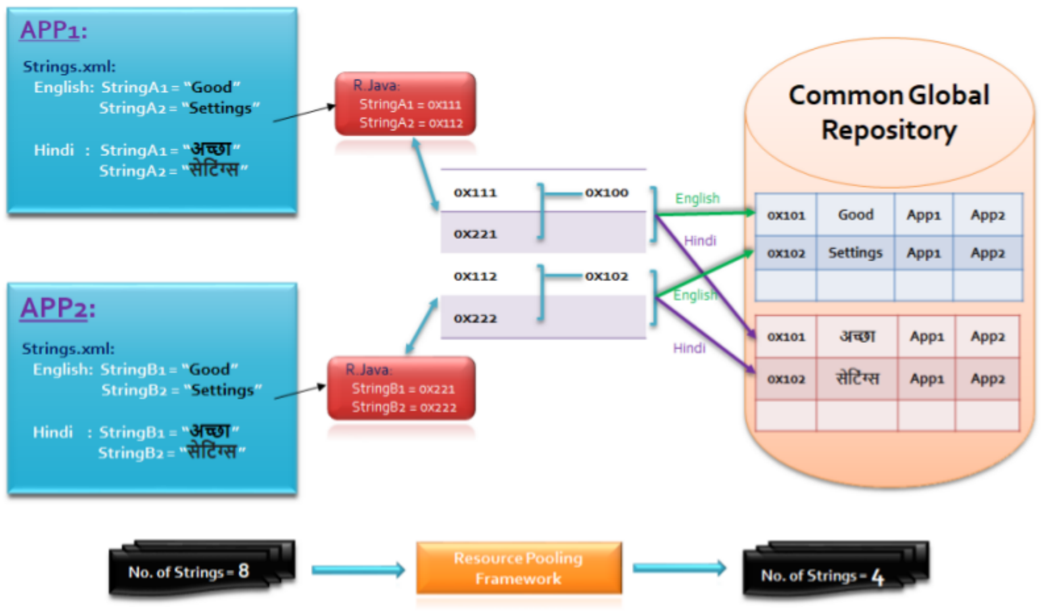


Fig. 4.1: Resource Pool Framework

When an application is uninstalled, the RPF will remove its package id from all the resources in the resource pool. If any resource is left without any package ids, it will be deleted from the resource pool. The mapping file will also be updated to have only valid mappings. When an application requests a resource using the app res id, the RPF will first check the PP value of this res id. If it is 01(for framework), the RPF will redirect this request to the Framework's resource table. If this value is 7f (for applications), The RPF will convert this to resource pool res id using the mapping created earlier. It will then redirect this request to the resource pool. Fetching resource from resource pool will be same as Android's current architecture [Fig. 3].

When an application is uninstalled, the RPF gets all the string resources in our resource pool associated with this app. If these strings contain only this app's package id (PP value) in its usage, RPF directly deletes them from the pool. For remaining strings, RPF removes the package id associated with those Strings in our resource pool. It then removes all the entries from the mapping HashMap where:

- Key belongs to this app's res id.
- Value belongs to any of the deleted string resource's pool res id.

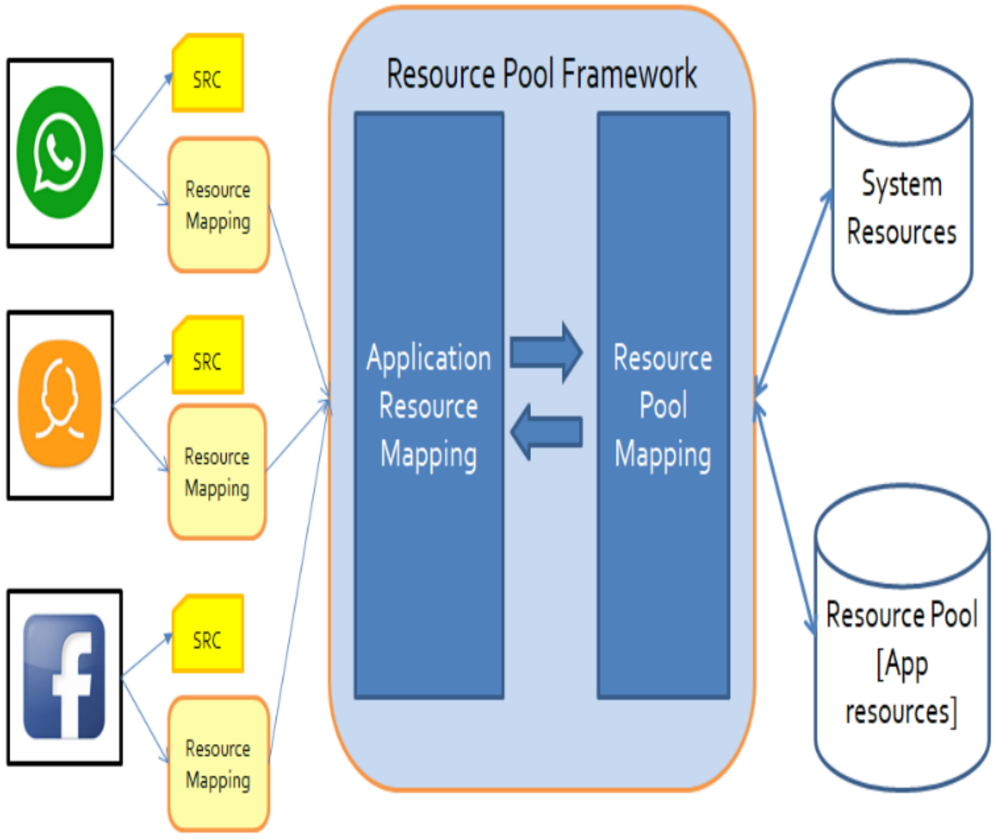


Fig. 4.2: Resource Pool Framework

Chapter 5: Results

The collective app size of all the pre-defined applications was 89.93MB [Table. I]. after applying the RPF solution, the collective size of resource pool and all apps without string resources came down to 77.97MB. This results in a saving of 13% in device storage [Table. II].

TABLE I: Apk size of pre-defined applications

S.No	**Application**	**App Size without Pool**	**App Without String Resource [A]**	**Pool Size [B]**
1	Contacts	8.84	4.63	**47.83**
2	Calendar	6.35	1.29	
3	Memory Saver	4.96	2.19	
4	Clock	6.72	2.2	
5	Keyboard	17.13	10.6	
6	Settings	45.93	9.23	
Total		**89.93**	**30.14**	

TABLE II: Memory saving in device storage

Total App Size Without Pool	**Total App Size With Pool [A] + [B]**	**Total Savings**
89.93	**77.97**	**13%**

Chapter 6: Conclusion

6.1 CONCLUSIVE RESULT:

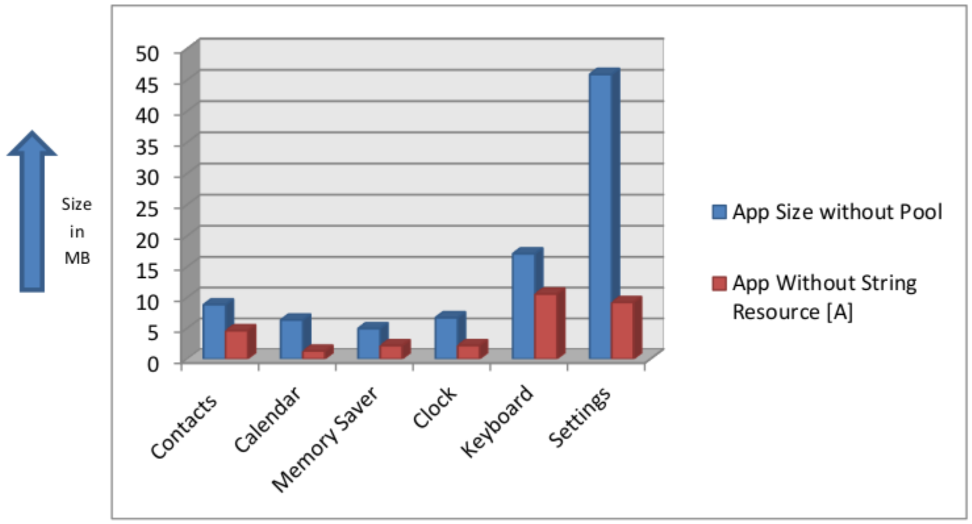


Fig. 6.1: Application size with and without resource

Through the above experiment we can see that the device storage can be improved with proposed architecture. Our experiment has saved about 13% of memory when applied on only 6 applications and string resources.

Once the solution will be applied to all the applications with all the resources like images (drawables which play a major role in apk size), layouts etc., it will have significant improvement

in storage. Through the RPF we can reduce/remove duplicate resource and provide more space to user for their personal data or to install new apps.

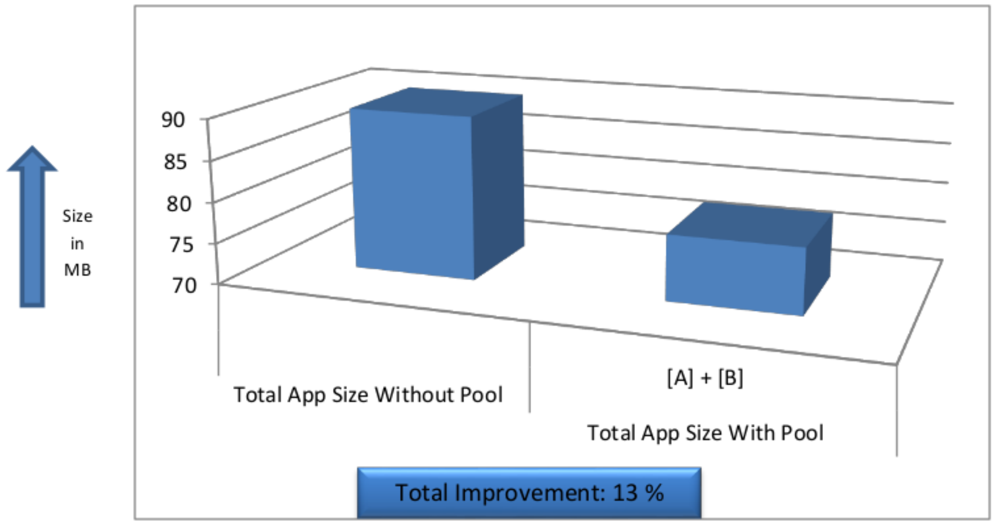


Fig. 6.2: Overall result

6.2 FUTURE WORK:

We have shown that the proposed solution improves device storage. It can be further enhance by moving this to cloud [Fig. 5]. A cloud based RPF will provide additional benefits as described below:

1 Common cloud-based platform enables sharing of larger pool of resources including Locale/ language.

2 Intelligent mechanism for fetching resources from pool to device based on

Device specific Images/Icon resources which are required for specific resolutions.

Only limited String resources which are required by installed applications.

3 Larger number of contributors/seeders enable greater accuracy of results.

4 Apply the solution to all applications including downloadable apps from market or play store.

5 It will also be enhanced to support all applications which do not support multi-language.

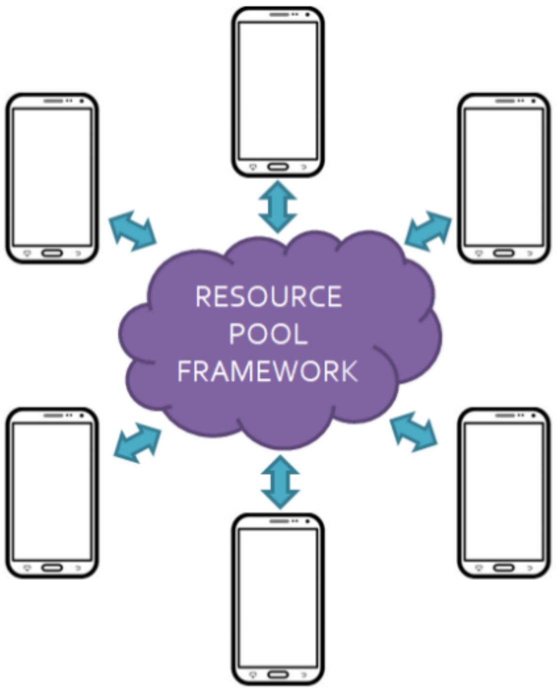


Fig. 6.3: The Cloud Based Resource Pool

REFERENCES

[1] Aapt2 : Android developers. https://developer. android.com/studio/command-line/aapt2. [Accessed on 05.10.2014].

[2] Android gradle plugin release notes : Android developers. https://developer.android.com/studio/releases/gradle-plugin. Accessed Oct 20, 2019].

[3] Android studio : Android developers. https://developer. android.com/studio. [Accessed Oct 20, 2019].

[4] App resources overview : Android developers. https://developer.android.com/guide/topics/resources/providing-resources. [Accessed Oct 20, 2019].

[5] Application fundamentals : Android developers. https://developer.android.com/guide/components/fundamentals.[Accessed Oct 20, 2019].

[6] Asset manager: Android class. https://developer.android.com/reference/android/content/res/AssetManager. [Accessed Oct 20, 2019].

[7] Gradle build tool. https://gradle.org/. [Accessed Oct 20,2019].

[8] Inside the android application framework - 2008 google i/o session videos and slides. https://sites.google.com/site/io/inside-the-android-application-framework. [AccessedOct 20, 2019].

[9] Overview of memory management : Android developers. https://developer.android.com/topic/performance/memory-overview#SharingRAM. [Accessed Oct 20,2019].

[10] Platform architecture : Android developers. https://developer.android.com/guide/platform. [Accessed Oct 20,2019].

[11] Pool (computer science). https://en.wikipedia.org/wiki/Pool (computer science). [Accessed Oct 20, 2019].

[12] Resource types overview : Android developers. https://developer.android.com/guide/topics/resources/available-resources.html. [Accessed Oct 20, 2019].

[13] Resources: Android class. https://developer.android.com/reference/android/content/res/Resources.html. [AccessedOct 20, 2019].

[14] Cong Li, Jia Bao, and Haitao Wang. Optimizing low memory killers for mobile devices using reinforcement learning. Pages 2169–2174, 06 2017.

[15] Rasmus. The zygote process. https://medium.com/masters-on-mobile/the-zygote-process-a5d4fc3503db,Mar 2015. [Accessed Oct 20, 2019].

[16] Kashif Tasneem, Ayesha Siddiqui, and Anum Liaquat.Android memory optimization. International Journal of Computer Applications, 182:36–43, 02 2019.

[17] Kumar Vimal and Aditya Trivedi. A memory management scheme for enhancing performance of applications on android. pages 162–166, 12 2015.

[18] Cheng-Zen Yang and Bo-Shiung Chi. Design of an intelligent memory reclamation service on android. Pages 97–102, 12 2013.

[19] Zhu Zongwei, Fan Wu, Jing Cao, Xi Li, and Gangyong Jia. A thread-oriented memory resource management framework for mobile edge computing. IEEE Access, 7:45881–45890, 01 2019.

swt 512_4

ORIGINALITY REPORT

5%	3%	1%	5%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Delhi Technological University Student Paper	3%
2	louisdl.louislibraries.org Internet Source	1%
3	ecommons.usask.ca Internet Source	<1%
4	Cong Li, Jia Bao, Haitao Wang. "Optimizing low memory killers for mobile devices using reinforcement learning", 2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC), 2017 Publication	<1%
5	Submitted to The Hong Kong Polytechnic University Student Paper	<1%
6	www.i-scholar.in Internet Source	<1%
7	Tzu-Hsiang Su, Hsiang-Jen Tsai, Keng-Hao Yang, Po-Chun Chang, Tien-Fu Chen, Yi-Ting	<1%

Zhao. "Reconfigurable vertical profiling framework for the android runtime system", ACM Transactions on Embedded Computing Systems, 2014

Publication

Exclude quotes	On	Exclude matches	Off
Exclude bibliography	On		