A Thesis

entitled

Optimizing Android Memory Management by Predicting User Behavior

by

Srinivas Muthu

Submitted to the Graduate Faculty as partial fulfillment of the requirements for the Masters of Science Degree in Engineering

Dr. Jackson Carvalho, Committee Chair

Dr. Mansoor Alam, Committee Member

Dr. Henry Ledgard, Committee Member

Dr. Patricia R. Komuniecki, Dean
College of Graduate Studies

The University of Toledo
December 2015

An Abstract of

Optimizing Android Memory Management
by Predicting User Behavior

by

Srinivas Muthu

Submitted to the Graduate Faculty as partial fulfillment of the requirements for the Masters of Science Degree in Engineering

The University of Toledo
December 2015

With the advent of increase in the amount of RAM (Random Access memory) available in Android smart-phones, there is a case to be made that this additional memory availability can be put to better use. By default, every Android application runs in its own Linux process. When a smart-phone that runs on Android is active, the RAM contains all the active processes and services (processes that run in the background). In addition to these processes and services, the RAM also contains cached background processes. These processes are kept in memory so that in the event the user clicks any of the corresponding applications, they can be loaded onto the screen quickly. If Android needs to reclaim memory for other processes, it eliminates cached background processes through an LRU (Least Recently Used) scheme. We postulate that analyzing user behavior could help us better determine which applications to cache in memory, as cached background processes. We parse the user's Calendar for contextual clues and gather information that could help us predict which application a user is about to use. We measure the cache efficiency (Cache hits and misses, CHR (Cache Hit Ratio)) for the default CRA (Cache Replacement Algorithm) Android employs (LRU), a pure prediction based CRA and finally a hybrid approach that combines the LRU approach with the prediction approach. We also demonstrate why the hybrid CRA is the most efficient one.

To my parents who've always put my needs ahead of theirs.

Acknowledgments

First and foremost, I'd like to thank Dr. Jackson Carvalho for mentoring me throughout my time here at UT. Without his guidance, I would'nt be half the student I am today. I'd like to thank Dr. Mansoor Alam for believing in me and offering me tuition scholarship to pursue graduate studies, here at UT. I'd like to thank Dr. Lawrence Thomas for being an exemplary professor and his words of wisdom have always guided me in tough times.

I am grateful to Dr. Henry Ledgard for taking the time to be on my committee and mentoring me during my freshman year. I'd like to thank Dr. Donald White and his students for helping me with data collection, design and representation. It was nothing short of a privelege to work with Dr. White and his students. I'd like to thank Dean Nagi G. Naganathan for employing me and guiding me over the course of my time in graduate school.

I'd like to thank all my friends and family for supporting me throughout my time here at UT but I'd like to especially mention Sandy Stewart for everything she's done for me. This page is'nt enough to expound on the details but it suffices to say I would'nt be here without her and she is like a second mother to me.

Contents

List of Tables

List of Figures

List of Abbreviations

RAM	Random Access Memory
LRU	Least Recently Used
CHR	Cache Hit Ratio
CRA	Cache Replacement Algorithm
OS	Operating System
AOSP	Android Open Source Project
APK	Android Package
IPC	Inter Process Communication
IC	Integrated Circuits
MB	Mega-Byte
GB	Giga-Byte
APC	Android Process Cache
APMD	Android Powered Mobile Device
CBP	Cached Background Processes
MRU	Most Recently Used
LFU	Least Frequently Used
RR	Random Replacement

List of Symbols

✓ ..... Represents a check mark indicating that a particular item has been checked or in a different context, whether the checked item is correct

Preface

This thesis is original, unpublished, independent work by the author, Srinivas Muthu under the tutelage of Dr. Jackson Carvalho.

Chapter 1

Introduction

1.1 Android Operating System

1.1.1 Overview of the Android OS

Android is a mobile OS (Operating System) based on the Linux kernel and designed primarily for touchscreen devices such as smart-phones and tablets [1]. In addition to touchscreen devices, Android TV, Android Auto and Android Wear are emerging technologies with specialized user interfaces. Globally, it is the most popular mobile OS [2]. Android has an active community of developers and enthusiasts who use the AOSP (Android Open Source Project) source code to develop and distribute their own modified versions of the operating system [4]. Android homescreens are typically made up of app icons and widgets. App icons launch the associated app, whereas widgets display live, auto-updating content such as the weather forecast, the user's email inbox, or a news ticker directly on the homescreen [5].

Internally, Android OS is built on top of a Linux kernel. On top of the Linux kernel, there are the middleware, libraries and APIs written in C and application software running on an application framework. Development of the Linux kernel continues independently of other Android's source code bases.

Figure 1-1: [3] Android System Architecture

1.1.2 Applications, Activities and Services

1.1.2.1 Applications

Android apps are written in the Java programming language. The Android SDK tools compiles the code, along with any data and resource files into an APK (Android Package), which is an archive file with an .apk suffix. One APK file contains all the contents of an Android app and is the file that Android-powered devices use to install the app [6]. The Android operating system is a multi-user Linux system in which each app is a different user. Each process has its own virtual machine (VM), so an app's code runs in isolation from other apps. By default, every app runs in its own Linux process.

1.1.2.2 Activities

An Activity is an application component that provides a screen with which users can interact in order to do something, such as dial the phone, take a photo, send an email, or view a map [7]. An application usually consists of multiple activities that are loosely bound to each other. Typically, one activity in an application is specified as the main activity, which is presented to the user when launching the application for the first time. Each activity can then start another activity in order to perform different actions.

1.1.3 Services

A Service is an application component that can perform long-running operations in the background and does not provide a user interface [8]. Another application component can start a service and it will continue to run in the background even if the user switches to another application. Additionally, a component can bind to a service to interact with it and even perform IPC (Inter Process Communication). For example, a service might handle network transactions, play music, perform file I/O, or interact with a content provider, all from the background.

1.2 Growth of available RAM over the years

1.2.1 Advances in Technology

RAM is a form of computer data storage. A RAM device allows data items to be accessed (read or written) in almost the same amount of time irrespective of the physical location of data inside the memory. The overall goal of using a RAM device is to obtain the highest possible average access performance while minimizing the total cost of the entire memory system. Today, random-access memory takes the

form of IC (Integrated Circuits)(s).

1.2.1.1 Moore's Law

Moore's law is the observation that the number of transistors in a dense IC doubles approximately every two years.

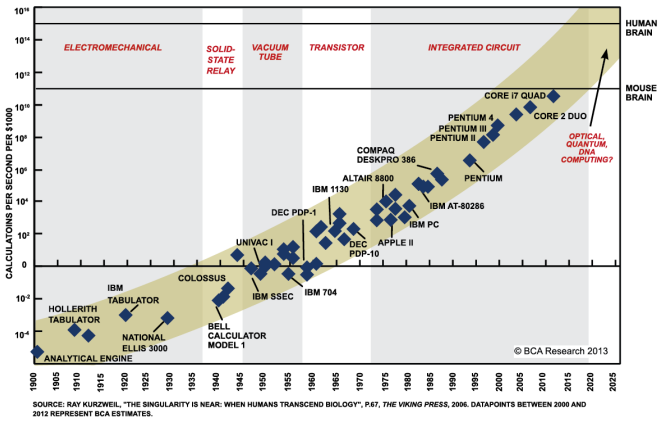

Figure 1-2: [9] Moore's Law

Advancements in digital electronics are strongly linked to Moore's law, especially in the context of memory capacity. T-Mobile G1, the very first Android smart-phone to be released back in 2008 [10] has a 256 MB (Mega-Byte) RAM [11]. In comparison, the Nexus 6P that was released in September 2015 has a 3 GB (Giga-Byte) RAM and some devices like the LG T585 have a 16 GB RAM [12][13], which amounts to 64 times the memory capacity of the T-Mobile G1. The rapid growth in the amount of RAM available to Android devices has in part led to newer possibilities and in our case, a better CRA for determining which processes (in the context of Android applications) remain in memory.

1.2.2 Demise of Task Killers

One of the main benefits of the Android OS is the fact that unlike certain other OS(s), it can run apps in the background. This enables us to have multiple applications open at the same time which results in true multitasking. Thus, RAM availability is highly desirable [14]. A popular misconception is that forcibly removing applications from memory in order to 'free up RAM' will result in increased performance. In fact, several task-killer applications promise to do precisely that. With the advancement in the amount of RAM available, the debate should be about how to better use all this extra space, not killing applications to 'free up' more space. In fact, the Android OS can go one step further and pro-actively cache applications that the user might use in the near future. We'll analyze this prospect in more detail in the upcoming chapters.

1.3 How Android Manages Processes

1.3.1 Android Process Lifecycle

A Android process can be in one of five different states at any given time, from most important to least important [15]:

- Foreground process: A foreground process is one that is required for what the user is currently doing.
- Visible process: A visible process is one holding an Activity that is visible to the user on-screen but not in the foreground.
- Service Process: Service processes are not directly visible to the user, they are generally doing things that the user cares about (such as playing music in the background).

- Background process: A background process is one holding an Activity that is not currently visible to the user. They are kept in an LRU list to ensure the process that was most recently seen by the user is the last to be killed when running low on memory.

- Empty process: An empty process is one that doesn't hold any active application components. The only reason to keep such a process around is as a cache to improve startup time the next time a component of its application needs to run.

Every Android smart-phone user is capable of checking the processes that currently reside in physical memory. The Application Manager which is part of the Settings app shows a list of running processes and cached background processes. Additionally, it breaks down the composition of RAM usage by the System applications and user installed applications.

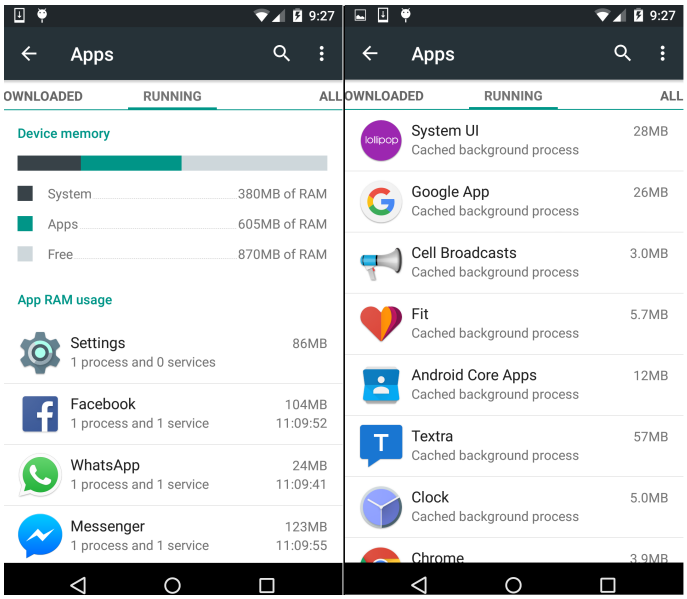

Figure 1-3: Left - List of Running Applications (Active Processes and Services)
Right - List of Cached Background Processes

1.3.2 LRU Cache

LRU is a CRA that discards the least recently used items first. This algorithm requires keeping track of what was used when, which is expensive if one wants to make sure the algorithm always discards the least recently used item [17]. Android keeps processes that are not hosting a foreground app component in a LRU cache. As the system runs low on memory, it may kill processes in the LRU cache beginning with the process least recently used, but also giving some consideration toward which processes are most memory intensive [16].

1.4 Goals and Objectives

1.4.1 A User-Centric CRA

The main benefit of caching processes (components of applications recently clicked by the user) in memory is to improve startup time (of the application), the next time a component of the application needs to run. This greatly enhances the usability experience and therefore it's in Android's best interests to increase the efficiency of this cache i.e. improve its CHR. In addition to recency of application usage (which is what Android currently relies on in the form of LRU as a CRA), there is scope to potentially infer what application a user may click in the near future. A more user-centric CRA could look into contextual clues and deduce patterns in user behavior to improve the decision making involved in determining which applications to cache in memory, pro-actively or otherwise. We will explore in theoretical detail many such possibilities for user behavior inference and as a proof of concept, read the user's Calendar application and parse for contextual information that could help in improving the CHR of the APC (Android Process Cache).

1.5 Organization of Thesis

Several works in the past have focused on context-aware applications and ways to observe user behavior patterns. We'll explore some these works and how this contextual data was put to use in Section 2. In Section 3, we'll take a look at some of the challenges in collecting cache metrics like detecting what's in the RAM, detecting when the user clicks a new application, whether to approach the problem at the user level or the system level, to list a few. Section 4 elaborates on the setup of the experiment and analyzes the design of the applications used to collect these metrics. We dive into the data in Section 5 and summarize its ramifications. We also investigate how certain factors could explain the variation in data. Section 6 talks about scope for future work and explores some of the possibilities of directly adding to this work. We conclude the thesis with Section 7.

Chapter 2

Related Work

2.1 Early Studies in Context-Aware Computing

Dey, Abowd and Salber, in their 2001 paper titled *A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications* laid out the foundation and in many ways a standardized framework for building context-aware applications. When this paper was published, mobile phones were becoming mainstream devices and were no longer confined to the hands of the wealthy. By definition, more people were mobile and the devices they carried with them had the potential to utilize contextual information gathered from the surroundings, in ways that desktops could'nt do due to their stationary nature. They defined context as any information that characterizes a situation related to the interaction between humans, applications and the surrounding environment. They opined that the state of research in context-awareness was inadequate for three main reasons:

- The notion of context was ill-defined.
- There was a lack of conceptual models and methods to help drive the design of context-aware applications.
- No tools were available to jump-start the development of context-aware appli-

cations.

They focused their efforts on the pieces of context that could be inferred automatically from sensors in a physical environment and produced Context Toolkit [20], a conceptual framework that supported the rapid development of context-aware applications. It is important to keep in mind that at the time this paper was published, the notion of what we refer to as a smart-phone was non-existent. This is evident from the fact that most APMD(s) have built-in sensors that measure motion, orientation, and various environmental conditions. These sensors are capable of providing raw data with high precision and accuracy, and are useful in monitoring three-dimensional device movement or positioning, or changes in the ambient environment near a device [21]. These conditions make APMD(s) very suitable for utilizing contextual information and translating that potential into a better user experience. We utilize this feature of APMD(s) to propose ways to improve user experience by incorporating contextual information in managing the physical memory of the APMD.

2.2 Context Patterns in Application Usage

Several works have attempted to collect user behavior data and they've accomplished this in unique ways. Hannu Verkasalo, in his 2007 paper titled *Contextual Patterns in Mobile Service Usage* analyzes how mobile services are used in different contexts [18]. Usage contexts were divided into *home, office* and *on the move*. A specialized algorithm tracked user's location patterns over a period of time to determine whether the user was at home or at work. Furthermore, changes in location data revealed whether the user was in transit between home and work. There was a correlation between the mobile services requested by the user and the user's physical location, which enabled the algorithm to understand the user's context quite well (refer Figure 2-1). It was also observed that certain services were requested more

during the weekends as opposed to certain others being requested more during the weekdays i.e. days with office hours.

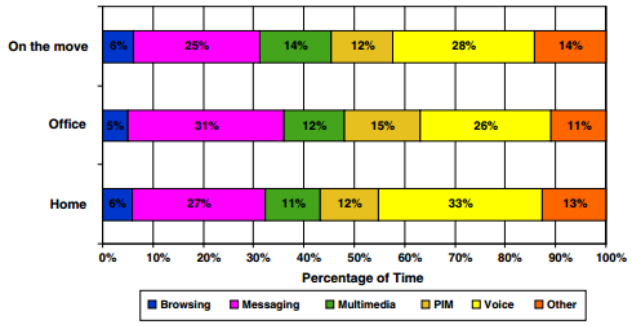

Figure 2-1: By classifying user context into three distinct entities namely *home, work* and *on the move*, the author managed to infer which services were requested more in a given context. For e.g. Voice services i.e. calling was predominantly requested while the user was at home. This type of information can be very useful.

2.3 Context Phone

Mika Raento, Antti Oulasvirta, Renaud Petit, and Hannu Toivonen, in their paper titled *ContextPhone: A Prototyping Platform for Context-Aware Mobile Applications* proposed that mobile phones are well suited for context aware computing due to an intimate relationship between the user and the phone [19]. Their primary goal was to provide context as a resource and in order to accomplish that, developed a *ContextPhone* which comprised of four components (Refer Figure 2-2):

- Sensors that acquire contextual data (such as location data from GPS)
- Communication services (such as SMS, MMS to list a few)
- Customizable Applications (that can replace existing applications)
- System Services (such as background services for error logging)

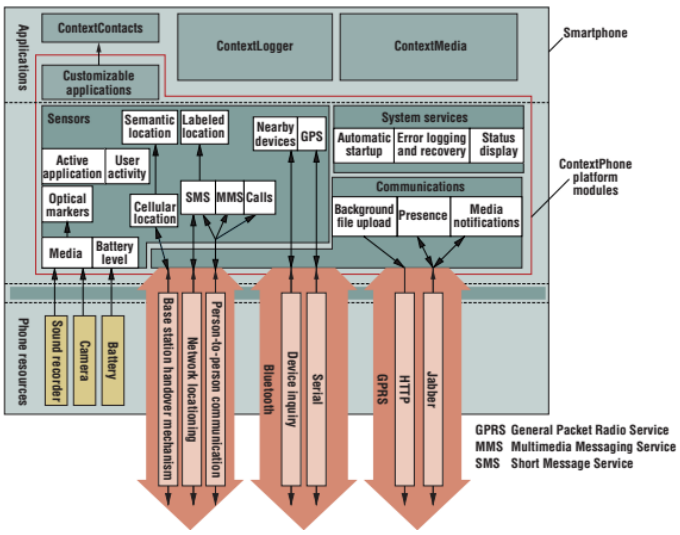

Figure 2-2: The ContextPhone platform; Four interconnected components sensors, system services, communication services, and customizable applications facilitate communication with the outside world.

It is important to observe that this paper came out in 2005, two years before the first version of Android was even launched. Yet, their fundamental point of focus is still relevant in how we can use context as a powerful resource. Their point about the necessity of customizable applications that are needed for context information to be relevant perfectly applies to this thesis. The Android OS which is open sourced as AOSP is as customizable as it can get when it comes to modifying mobile components and by extension the user experience with APMD(Android Powered Mobile Device)(s). To be specific, it facilitates ways to gather cache related information such as the list of processes currently residing in memory, the current foreground application and whether the caching mechanism per se is up for modification to list a few. It would be impossible to collect and/or modify such information on any other mobile OS (such as Apple's iOS).

2.4 Using Context Information for Authentication

Dey et al. established a framework for collecting contextual data and Verkasalo and Raento et al. have shown how user behavior can be analyzed through sensors and other modules that gather contextual information. Shi et al.'s paper titled *Implicit Authentication through Learning User Behavior* demonstrates a similar approach in collecting behavioral data of the user but differ in how they put that information to use. They have devised a model to implicitly authenticate smart-phone users based on their behavioral patterns. The user's usage patterns are collected (for e.g. how many calls a user makes a day, how many times user's location changes (Refer Figure 2-3) etc.) and this information is used in building a user model [22].

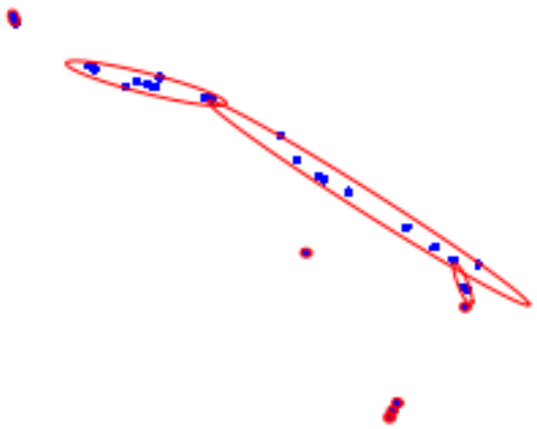

Figure 2-3: The blue dots represent the users traces in a two-hour epoch over multiple days, and the red ellipses represent the clusters fitted. The major two directional clusters correspond to the users trajectory on a highway

Once the user model is built and the profile completed, the algorithm can determine the likelihood of a random user of the phone not being the owner of the phone (the one whose user model was profiled).

As demonstrated by Shi et al., there are numerous ways contextual information can come in handy. In our case, we propose a user-centric CRA for the APC, demonstrate

its superior efficiency and user experience, by looking into the user's calendar for contextual information.

Chapter 3

Challenges in Collecting the Desired Metrics

3.1 Introduction

Now that we've laid the foundation for using contextual information to influence which processes Android caches in memory, we need to establish the metrics that need to be collected in order to measure the efficiency of the existing CRA and the proposed CRA. The following section addresses the metrics we need.

3.2 What are the desired metrics?

3.2.1 Hits, Misses, Hit Ratio & Latency

Alan Jay Smith defines Cache as a high speed buffer that holds items in current use [23]. In our context, Android holds application components (such as processes) as CBP (Cached Background Process)(s) in memory. As previously discussed, CBP(s) do not take up processor time, they are merely cached for quick application startup should the user request that particular application. A cache hit is a state in which data requested for processing by a component or application is found in the cache

memory. It is a faster means of delivering data to the processor, as the cache already contains the requested data [24]. When applied to our scenario, a cache hit would represent a situation wherein the user requests (clicks) an application that currently resides in memory, either as an active process (including services) or as a CBP (Refer Figure 3-1).

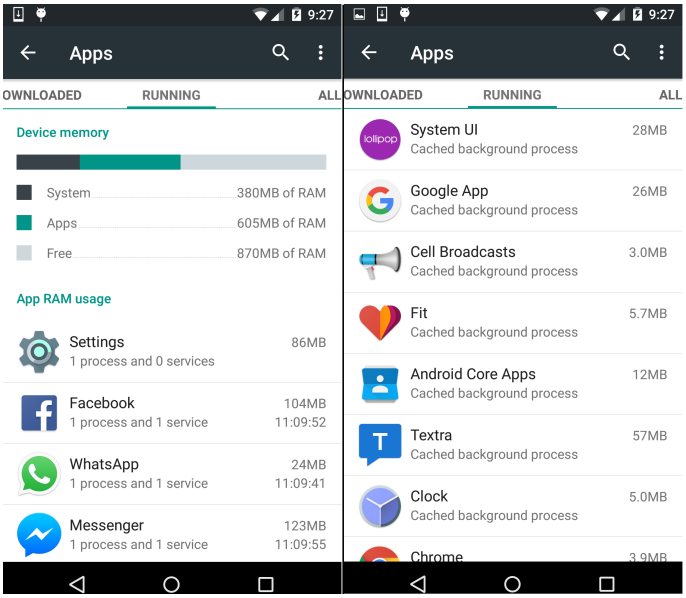

Figure 3-1: Given this snapshot of the RAM, if the user clicks on the Messenger application (active process) or Clock application (CBP), it would result in a Cache Hit

A cache miss is a state where the data requested for processing, by a component or application is not found in the cache memory [25]. When applied to our scenario, a cache miss would represent a situation wherein the user requests (clicks) an application that currently does not reside in memory, either as an active process (including services) or as a CBP (Refer Figure 3-2).

CHR is defined as the percentage of cache hits compared to the overall number of requests (total of cache hits and cache misses).

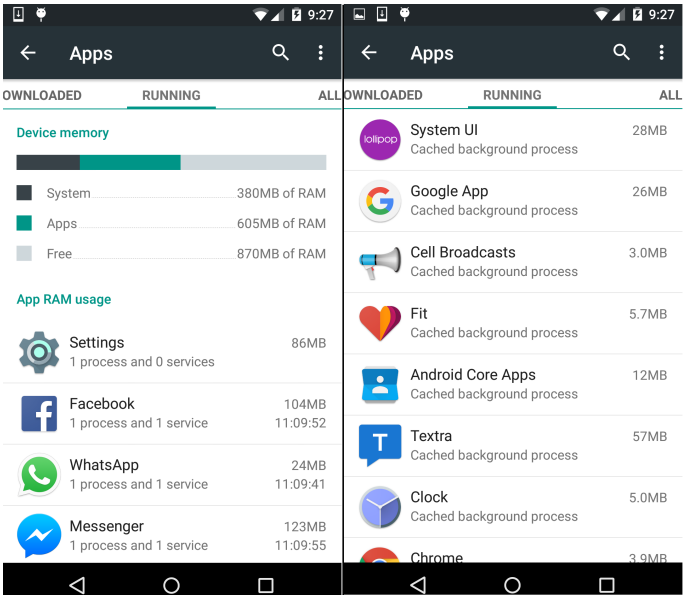

Figure 3-2: Given this snapshot of the RAM, if the user clicks on say, the Google Drive application (process not in memory), it would result in a Cache Miss

$$OverallNumberOfRequests = CacheHits + CacheMisses \tag{3.1}$$

$$CHR = \frac{CacheHits}{OverallNumberOfRequests} * 100 \tag{3.2}$$

There are two primary figures of merit of a cache [23]:

- Latency
- CHR

The latency of a cache describes how long after requesting a desired item, the cache can return that item (in the case of a cache hit). In our scenario, processes are cached in RAM. Therefore, even if the number of processes cached in memory increases, there will hardly be any difference in the time it takes to fetch a reque-sted item (application component) as they all reside in RAM which by definition supports

random access. Latency is more of a factor in caches that are located further away from physical memory (such as L1 and L2 caches).

The CHR of a cache describes how often a searched-for item is actually found in the cache. The higher the CHR, the better the cache is. When applied to our context, a high CHR results is more application components being fetched from memory, rather than disk which in turn results in quicker application startup times. This improvent in startup speed results in an enhanced user experience.

3.2.2 CRA(s) Under Consideration

Now that we've determined the fundamental metrics we need, let's analyze the various CRA(s) including the one Android currently uses, namely LRU.

3.2.2.1 Optimal CRA

The most efficient CRA would be to always discard the information that will not be needed for the longest time in the future. In our case, the ideal CRA would be one that ensures that every application the user requests, resides in memory. This optimal result is referred to as Belady's optimal algorithm [26] or the clairvoyant algorithm. Since it is generally impossible to predict how far in the future information will be needed, this is not implementable in practice. The practical minimum can be calculated only after experimentation, and one can compare the effectiveness of the actually chosen CRA against Belady's as a benchmark.

3.2.2.2 LRU

This is the default CRA Android currently uses to discard processes from memory. LRU is a type of CRA that discards the least recently used items first. This algorithm keeps track of what was used when, and it's expensive to ensure that the least recently used item is always discarded first. When applied to our scenario, Android caches

application components in memory as CBP(s) and associates each with a timestamp. If the memory gets too full, it starts removing the least frequently used CBP(s) first and depending on how direly low the memory is, it may not stop killing processes until even the active foreground application (the one user is interacting with) is removed, [15] although this situation is extremely rare in practice and only results in the case of malfunctioning applications that leak memory. The Settings application can give the user clues about which CBP(s) were recently used. Clicking on an application that is currently not in memory results in that application being cached as a CBP near the top of the list. This suggests that the CBP(s) displayed by the Settings application are roughly sorted by their timestamp from MRU to LRU (Refer Figure 3-3).

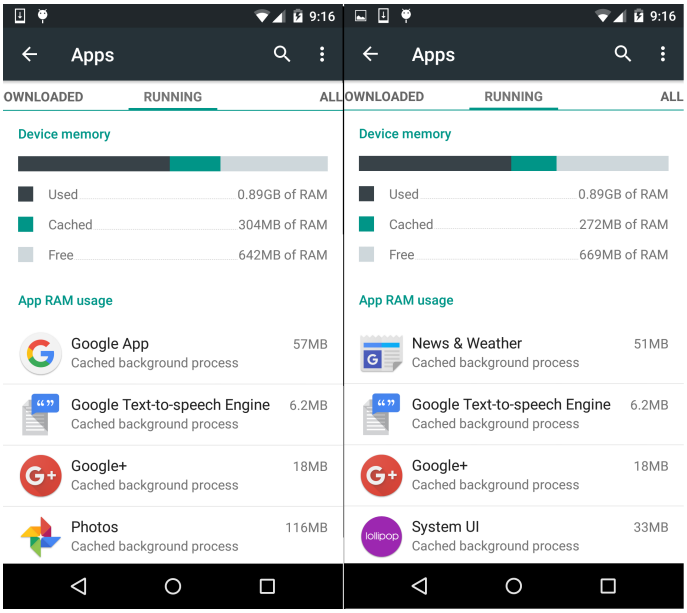

Figure 3-3: The image to the left describes the snapshot of the CBP(s) as displayed to the end-user by the Settings app. The image to the right displays the snapshot of the CBP(s) after the end-user requests for the News and Weather application. Note that the News and Weather application will not be cached as a CBP until the user is done interacting with it as up until that point, it will reside in memory as an active process.

It is worth noting that a perfect implementation of LRU requires a timestamp on

each reference, and the OS needs to keep a list of items ordered by the timestamp. This implementation may be deemed too expensive in certain cases especially if the frequency of memory references is high. The common practice is to approximate the LRU behavior and this can be achieved in other ways [27].

3.2.2.3 Most Recently Used (MRU)

MRU discards, in contrast to LRU, the most recently used items first. In findings presented at the 11th VLDB conference, Chou and DeWitt noted that when a file is being repeatedly scanned in a looping sequential reference pattern, MRU was the best replacement algorithm [28]. Subsequently other researchers presenting at the 22nd VLDB conference noted that for random access patterns and repeated scans over large datasets (sometimes known as cyclic access patterns) MRU cache algorithms had more hits than LRU due to their tendency to retain older data [29]. In our scenario, CBP(s) neither comprise a 'large dataset' nor are they repeatedly scanned in a looping sequential reference pattern (as they are only invoked during application requests which are on average, not that often), therefore it's unlikely that MRU would result in a high CHR but it's worth mentioning nonetheless. MRU algorithms are most useful in situations where the older an item is, the more likely it is to be accessed, which again is not a feature of CBP(s).

3.2.2.4 Least Frequently Used (LFU)

LFU is a type of CRA usually associated with memory management within a computer. The standard characteristics of this method involve the system keeping track of the number of times an item is referenced in memory. When the cache is full and requires more room the system will purge the item with the lowest reference frequency. This could be a really good fit with our scenario given the tendency of most smart-phone users to stick with the same set of popular applications and

use them in rotation [31]. Under these circumstances, the frequency of a select few applications would be really high and would always be cached in memory as they're requested often. LFU is sometimes combined with a Least Recently Used algorithm called LRFU [30]. We'll explore some of the possibilities with LFU in the future scope section and in our proposed hybrid approach where we incorporate global frequency of application usage in obtaining contextual information.

3.2.2.5 Random Replacement (RR)

Much like the name suggests RR is a CRA that randomly selects a candidate item and discards it to make space when necessary. This algorithm does not require keeping any information about the access history. For its simplicity, it has been used in ARM processors [32]. When applied to our scenario, it does'nt seem like a viable option, mainly because there is'nt a benefit in randomly caching applications in memory.

3.2.2.6 LRU - Context Hybrid

So far we've analyzed the standard CRA(s) that focus on discarding items that were either accessed least recently, least frequently or in other standard ways. Our approach combines the applications that Android caches through an LRU scheme with a separate context-analyzer module that reads the user's calendar and predicts which applications the user is likely to use. When the user requests an application, the combined set of applications (default applications in LRU and the list of suggested applications) is used as the base to predict whether a cache hit or cache miss occurred. We'll analyze in detail how each module functions but before that, let's take a look at whether Android caches CBP(s) pro-actively and how applications will be removed from memory in the cases where it runs too low.

Pro-actively Adding CBP(s) In order for the proposed LRU-Context Hybrid (henceforth referred to as the Hybrid approach) CRA to work, there must be provision for the Android OS to pro-actively cache application components in memory as CBP(s). Presuming that the context-analyzer module does its job and suggests a list of applications the user might use in the near future (based on user's Calendar information), there must a mechanism for Android to utilize that information and pro-actively cache these applications in RAM as CBP(s). This mechanism can be verified at the user level through a small experiment. As previously mentioned, the Settings application provides a visual interface for the list of active processes and services running in memory and the CBP(s) that reside in memory at any moment in time. As part of the experiment, each and every CBP was forcefully removed from the cache (there is provision to do this at the user level (Refer Figure 3-4)).

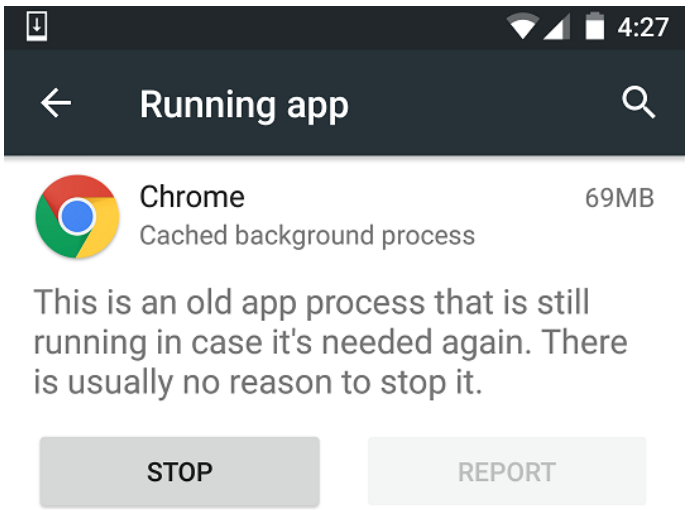

Figure 3-4: Pressing the *STOP* button removes the CBP from memory

Once each CBP was removed, the phone was untouched for three minutes. After three minutes, when the Settings application was launched to look at the list of CBP(s), it wasn't empty. Instead, there were several CBP(s), some of which were previously removed and some of which were'nt in the list to begin with (Refer Figure

3-5).

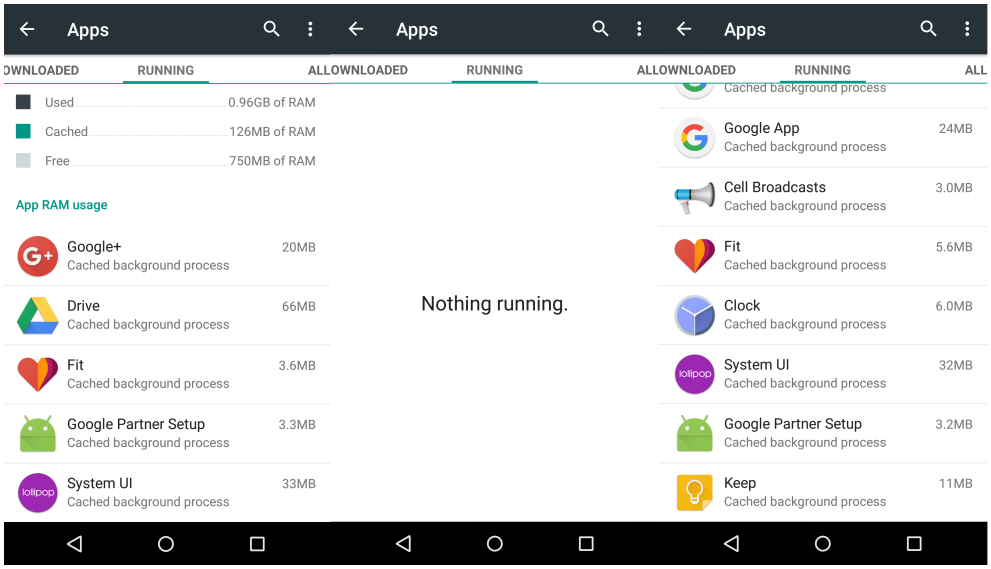

Figure 3-5: The leftmost image shows the snapshot of the CBP(s) before forceful deletion. After each and every CBP was removed manually, the screen displayed is shown in the middle image. The image farthest to the right shows the repopulated list of CBP(s), 3 minutes after the middle image was taken. This demonstrates Android's ability to utilize free RAM space and pro-actively cache CBP(s)

This suggestes that Android recognizes that an under-utilized RAM is a waste of resource and pro-actively caches applications that were'nt in memory to begin with, as CBP(s). This is also the reason why task killer applications fell out of favor [33]. This demonstrates that not only does Android provide a mechanism to pro-actively cache applications in memory, it already does so. The applications suggested by the Hybrid CRA approach can be introduced into the memory as CBP(s) in the same way. This is discussed in future scope section.

Prioritizing Overall Cache The Hybrid CRA would involve storing as CBP(s), both default application components cached by Android's LRU scheme and the application components of the list of applications obtained from analyzing the user's

Calendar. So the question now becomes, who gets kicked out when memory is running low? Since the applications inferred from user's contextual information do not have a recency associated with them, the LRU CRA cannot be applied over the overall list of CBP(s). A simple solution is to prioritize the CBP(s) obtained from the list of suggested applications over the default LRU based CBP(s) i.e. all the default CBP(s) are removed from memory before the CBP(s) cached as a result of context inference. There are two advantages to this approach:

- Only few applications are suggested by the context module.
  - For instance, say the user has a Calendar entry that reads *Call Mom*. There will be two or at most three applications suggested by the context-module that would be potentially useful to the smart-phone user in the context of calling his mother. This makes it more convenient to target CBP(s) cached by the LRU scheme in case of low memory.
- The probability of user requesting an application related to the Calendar event is quite high.
  - This is due to the fact that a recorded event in a user's Calendar is highly likely to occur and the applications needed in this context have a higher chance of being utilized as opposed to CBP(s) cached by the LRU scheme.
  - This is explored further in the future scope section where we discuss the relative probabilities of applications being used in conjunction with a Calendar event.

3.2.3 Supplementary Data

We've talked about the fundamental metrics involved in measuring the efficiency of the APC, namely cache hits, cache misses and CHR. We explored potential candidates

for serving as the CRA in managing the APC. There are other subtle measurements that are'nt as significant as the fundamental metrics but are important nonetheless as they shed light on certain attributes of the experiment. They are listed below:

- Number of Installed Applications
- Number of Unique Applications Requested (Clicked)
- Phone Model
- Android OS Version

Depending on the value of the overall number of installed applications and the overall number of unique applications requested by the end-user, certain CRA(s) have an edge over the others. Phone model information is collected to see if there is any correlation between models and cache efficiencies. Some of the techniques used to gather context information are not possible in certain Android OS versions. We'll analyze the correlation between these fields and the efficiency of CRA(s) in the Section 5.

3.3 Challenges

Now that we've established what the desired metrics are, let's examine some of the challenges in collecting them.

3.3.1 Deciding Between System vs User Level Approach

Firstly, we need to determine whether to approach the problem from a user level perspective or not. Attempting to solve the problem i.e. collecting the aforementioned metrics at the user level would involve building Android applications to gather relevant data. On the other hand, approaching the problem from a system level

would involve altering the source code of the Android OS (facilitated by AOSP) and recompiling it to build our own custom distribution, much like the CyanogenMod community [34]. It is easier to solve the problem (if possible) at the user level for two reasons:

- The complexity involved in altering OS source code is significantly higher compared to writing Android applications.
- It would be significantly harder to convince potential research volunteers to reboot their smart-phones with a custom distribution of the Android OS as opposed to asking them to install an Android application or two.

Given that it's beneficial to gather relevant data at the user level, we need to identify the complexities involved and determine whether they are solvable at the user level. In order to effectively determine which CRA is ideal for the APC, we need to identify whether the following problems can be approached from a user level perspective:

- Getting the list of processes currently in memory ✓
- Knowing when a new process is launched by the user ✓
- Reading the User's Calendar ✓

We'll inspect in detail, why and how we need to solve these problems (and others) in the next section but the '✓' indicates that they are indeed solvable at the user level and do not require system level changes.

3.3.2 Getting Processes in Memory

3.3.3 Detecting a Change in the Foreground Application

3.3.4 Reading the User's Calendar

3.3.5 Parsing the Calendar Information

3.3.6 Addressing Privacy Concerns

Chapter 4

Experiment Setup and Application Design

4.1 Introduction

4.2 Eligibility for Volunteers

4.3 Process and Duration of Experiment

4.4 Cache Analyzer Design

4.5 Context Analyzer Design

4.6 Tools, Version Control and APK

4.6.1 Android Studio

4.6.2 Git

4.6.3 Third Party APK(s)

Chapter 5

Data Analysis and Results

5.1 Introduction

5.2 Phase A Data

5.3 Phase B Data

5.4 Default Approach Metrics

5.5 Pure Prediction Approach Metrics

5.6 Hybrid Approach Metrics

5.7 Factors Influencing Variation In Data

5.7.1 Android Version and Phone Model

5.7.2 Number of Applications Installed

5.7.3 Number of Applications Used

5.7.4 User Bias

5.7.5 User Demographics

Chapter 6

Scope for Future Work

6.1 Introduction

6.2 Improving Context Analysis

6.2.1 Machine Learning in Calendar Parsing

6.2.2 Calendar Parser - LFU Hybrid

6.2.3 Other Ways of Gathering User Behavior Data

6.2.4 Custom Priorities in Hybrid Cache

6.3 Alternate Ways of Using the Contextual Information

6.3.1 Switching to Silent Mode

6.3.2 Disabling Texting at High Speeds

6.4 Implementing the Hybrid Cache

Chapter 7

Conclusion

References

[**1**] "The Android Source Code"
http://source.android.com/source/index.html, October 2015. Online; Last Accessed: October 22, 2015.

[**2**] "The Most Popular Smartphone Operating Systems Globally,"
http://www.lifehacker.com.au/2014/09/the-most-popular-smartphone-operating-systems-around-the-world/, September 2014. Online; Last Accessed: October 22, 2015.

[**3**] "Android System Architecture"
https://mahalelabs.files.wordpress.com/2013/03/androidstack-1.jpg, March 2013. Online; Last Accessed: October 22, 2015.

[**4**] "Best custom ROMs for the Samsung Galaxy S2"
http://www.cnet.com/uk/how-to/best-custom-roms-for-the-samsung-galaxy-s2/, April 2012. Online; Last Accessed: October 22, 2015.

[**5**] "Widgets — Android Developers"
http://developer.android.com/design/patterns/widgets.html, October 2015. Online; Last Accessed: October 22, 2015.

[**6**] "Application Fundamentals — Android Developers"
http://developer.android.com/guide/components/fundamentals.html, October 2015. Online; Last Accessed: October 22, 2015.

[**7**] "Activities — Android Developers"
http://developer.android.com/guide/components/activities.html, October 2015. Online; Last Accessed: October 22, 2015.

[**8**] "Services — Android Developers"
http://developer.android.com/guide/components/services.html, October 2015. Online; Last Accessed: October 22, 2015.

[**9**] "Wikimedia Upload"

http://www.extremetech.com/wp-content/uploads/2015/04/MooresLaw2.png, October 2015. Online; Last Accessed: October 22, 2015.

[**10**] "A Brief History of Android Phones"
http://www.cnet.com/news/a-brief-history-of-android-phones/, August 2011. Online; Last Accessed: October 22, 2015.

[**11**] "GSM Arena"
http://www.gsmarena.com/t_mobile_g1-2533.php, October 2008. Online; Last Accessed: October 22, 2015.

[**12**] "Nexus 6P"
http://www.androidcentral.com/nexus-6p, October 2015. Online; Last Accessed: October 22, 2015.

[**13**] "Phone Egg — LG T585"
http://us.phonegg.com/phone/4783-LG-T585, November 2013. Online; Last Accessed: October 22, 2015.

[**14**] "Android PSA: Stop Using Task Killer Apps"
http://phandroid.com/2011/06/16/android-psa-stop-using-task-killer-apps-now/, June 2011. Online; Last Accessed: October 22, 2015.

[**15**] "Processes and Application Life Cycle"
http://developer.android.com/guide/topics/processes/process-lifecycle.html, October 2015. Online; Last Accessed: October 22, 2015.

[**16**] "Managing Your App's Memory"
http://developer.android.com/training/articles/memory.html, October 2015. Online; Last Accessed: October 22, 2015.

[**17**] "Theodore Johnson, Dennis Shasha - A Low Overhead High Performance Buffer Management Replacement Algorithm"
http://www.vldb.org/conf/1994/P439.PDF, January 1994. Online; Last Accessed: October 22, 2015.

[**18**] "Hannu Verkasalo - Contextual patterns in mobile service usage"
http://lib.tkk.fi/Diss/2009/isbn9789512298440/isbn9789512298440.pdf, March 2008. Online; Last Accessed: October 23, 2015.

[**19**] "Mika Raento, Antti Oulasvirta, Renaud Petit, and Hannu Toivonen - ContextPhone: A Prototyping Platform for Context-Aware Mobile Applications"

http://dl.acm.org/citation.cfm?id=1070628, May 2005. Online; Last Accessed: October 23, 2015.

[20] "Anind K. Dey, Gregory D. Abowd and Daniel Salber - A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications "
http://www.cc.gatech.edu/fce/ctk/pubs/HCIJ16.pdf, January 2001. Online; Last Accessed: October 23, 2015.

[21] "Sensors — Android Developers"
http://developer.android.com/guide/topics/sensors/sensors_overview.html, October 2015. Online; Last Accessed: October 23, 2015.

[22] "Elaine Shi, Yuan Niu, Markus Jakobsson, and Richard Chow - Implicit Authentication through Learning User Behavior"
markus-jakobsson.com/wp-content/uploads/2010/11/ShiNiuJakobssonChow2010.pdf, November 2010. Online; Last Accessed: October 23, 2015.

[23] "Alan Jay Smith - Design of CPU Cache Memories"
http://www.eecs.berkeley.edu/Pubs/TechRpts/1987/CSD-87-357.pdf, August 1987. Online; Last Accessed: October 23, 2015.

[24] "Cache Hit — Techopedia"
https://www.techopedia.com/definition/6306/cache-hit, October 2015. Online; Last Accessed: October 23, 2015.

[25] "Cache Miss — Techopedia"
https://www.techopedia.com/definition/6308/cache-miss, October 2015. Online; Last Accessed: October 23, 2015.

[26] "Najeeb A. Al-Samarraie - And Page Replacement Algorithms: Anomaly Cases"
http://www.iasj.net/iasj?func=fulltext&aId=50712, January 2003. Online; Last Accessed: October 23, 2015.

[27] "LRU Replacement Policy"
https://www.seas.upenn.edu/ cit595/cit595s10/handouts/LRUreplacementpolicy.pdf, October 2015. Online; Last Accessed: October 24, 2015.

[28] "Hong-Tai Chou, David J. Dewitt - An Evaluation of Buffer Management Strategies for Relational Database Systems"
http://www.vldb.org/conf/1985/P127.PDF, October 1985. Online; Last Accessed: October 24, 2015.

[29] "Shaul Dar, Michael J. Franklin, Bjorn T. Jonsson, Divesh Srivastava, Michael Tan - Semantic Data Caching and Replacement"
http://www.vldb.org/conf/1996/P330.PDF, October 1996. Online; Last Accessed: October 24, 2015.

[30] "Donghee Lee, Member, IEEE, Jongmoo Choi, Member, IEEE, Jong-Hun Kim, Member, IEEE, Sam H. Noh, Member, IEEE, Sang Lyul Min, Member, IEEE, Yookun Cho, Member, IEEE, and Chong Sang Kim, Senior Member, IEEE - LRFU: A Spectrum of Policies that Subsumes the Least Recently Used and Least Frequently Used Policies"
http://u.cs.biu.ac.il/ wiseman/2os/lru/lrfu.pdf, December 2001. Online; Last Accessed: October 24, 2015.

[31] "An Upper Limit For Apps? New Data Suggests Consumers Only Use Around Two Dozen Apps Per Month"
http://techcrunch.com/2014/07/01/an-upper-limit-for-apps-new-data-suggests-consumers-only-use-around-two-dozen-apps-per-month/, July 2014. Online; Last Accessed: October 24, 2015.

[32] "ARM Documentation"
http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.set.cortexr/index.html, July 2014. Online; Last Accessed: October 24, 2015.

[33] "Does Your Android Device Really Need a Task Killer?"
http://www.technorms.com/40664/android-task-killer-apps, September 2014. Online; Last Accessed: October 24, 2015.

[34] "CyanogenMod"
http://www.cyanogenmod.org/, October 2015. Online; Last Accessed: October 24, 2015.