Introduction to Android ART; The next generation of Android Runtime

Thesis · June 2014

DOI: 10.13140/RG.2.2.16616.06406

1 author:

Laban Ndwaru

Strathmore University

2 PUBLICATIONS **0** CITATIONS

SEE PROFILE

Introduction to Android ART; The next generation of Android Runtime.

Laban Ndwaru — S13/20031/10 — lndwaru@egerton.ac.ke
Department of Computer Science
Egerton University

Abstract—The advancement in mobile technology has spurred developments on the various mobile platforms. The growth in technology has led to proliferation of fast and graphics optimized mobile devices. Android, an Open Handset Allience project has experienced alot of changes since its inception. Optimization and speed being a key goal to it's development.
Android OS runs on Dalvik runtime machine. Being a virtual machine, the runtime makes the Android OS slow. Android is also laggy due to it's multitasking feature. There have been attempts in the past to speeden up Dalvik but they failed. This posed a challenge to increasing Android's speed and optimization. Recently Google released Android 4.4, with it was the debut of the Android Runtime (ART) which replaces the Dalvik Virtual Machine.ART which strikes a good balance between portability and performance makes a device fast and further enhances graphics optimization. ART keeps a pre-compiled code for the Android applications, this was not possible with Dalvik VM. Many ART's experimental benchmarks now reveal 2x speed improvement on Android OS.
This paper reveals how the two runtimes work, checks on the advantages and drawbacks of ART over the current Dalvik runtime and later suggests on the feasibilty of ART as the next Google Android runtime project.

Index Terms—ART, Dalvik, Android,OS

1 Introduction

ART is the new Android runtime introduced recently with the release of Android 4.4. The runtime can be turned on optionaly on the Android device. This is available for the purpose of obtaining early developer and partner feedback.Though not fully endorsed in the android market, it contain several features which qualify it's feasibility.[1]

Dalvik runtime is the default runtime used by applications and some system services on Android.This default run-time was created for the android project.From the recent developments, Android has proved to be evolving its runtime from dalvik to ART. A few commits in Google Git indicate that Google has already started the process of transforming the OS.The change targets to transform the android OS to a 64bit OS. All this based on enhancing the speed and optimization on the android devices. Android 64bit OS will be compatible with newest ARM processors as compared to Apple which already released a product with the 64-bit ARMv8 instruction set.Intel recently announced that the new 64 bit android OS would support their new Atom Bay Trail CPUs.Android Inc. recently acquired Flexycore Inc. the company behind the creation of the 10X speed boost android app .[7]The company joined Google to improve in the creation of Android ART.In view of the latest developments, ART is a looks like a near replacement to the Dalvik VM.

This paper highlights the strengths and the drawbacks of ART, the proposed runtime in Android. I compare the algorithim behind the JIT(Just-in-Time) compilation in Dalvik runtime and AOT (Ahead-Of-Time) compilation in ART.I also include the measures that can also be placed to enhance ART, thus reduce the weight of drawbacks over its advantages.

2 Android, a laggy OS

Android is described as a laggy operating system due to its massive multitasking nature. Unlike Windows and iOs, Android does not have the pre-compiled App binaries. This is because it does not have a single unified runtime. This makes the Android OS slow with poor graphics optimization less compared to other platforms.

Recently Android released ART (Android Runtime), a project aimed at enhancing and optimizing the Android OS. ART which is written in C/C++ will replace the approach of compilation of the Dalvik runtime.ART enhances pre-compiled code. This takes less time to run on start-up. The process is still cumbersome and not so effecient. The ART is still not compactible with other Android applications.

3 Dalvik Virtual Machine

Dalvik VM is the default runtime machine in android OS.Dalvik runs applications and code written in Java.A standard Java compiler turns source code written as text files into Bytecode, then they are compiled into a .dex file that are Dalvik VM compatible.During compilation, java class files are converted into .dex files which are like a jar file if one were using the standard Java VM. They are then read and executed by the Dalvik Virtual Machine.In essense, Dalvik is a mobile-optimized version of a Java Virtual Machine, which is open-source project that runs better than a standard Java VM on our limited hardware based on memory size and processor speed.

3.1 Operations of Dalvik Virtual Machine

This is a Just-in-time (JIT) compiler that runs on:

1) Slow CPU
2) Relatively little RAM
3) OS with no swap space
4) Powered by a battery

The Dalvik virtual machine uses register-based architecture.The figure 1 below shows the android kernel which composes of the Dalvik virtual machine in the android's runtime.It is a processing virtual machine, whereby the the underlying Linux kernel of the Android OS creates a new instance of Dalvik VM for every process running. This reduces the chances of multi-application failure if one Dalvik VM crashes.

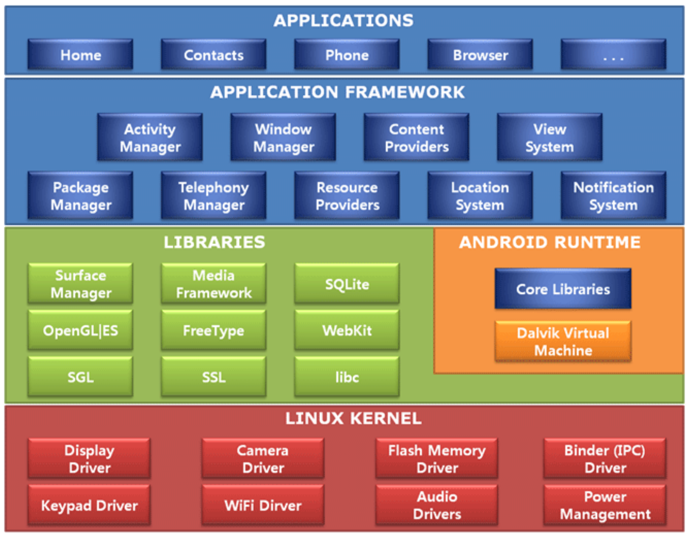


Fig. 1. Android's Kernel

Dalvik VM uses a 16 bit instruction set.The registers are implemented in Dalvik as 4 bit fields. When the system boots up, the boot loader loads the kernel into memory and initializes system parameters. Afterwards, a few processes takes place:

The kernel runs the Init program-This is the parent process for all processes in the system. The Init program starts system deamons and the very important Zygote service. The Zygote process creates a Dalvik instance.This becomes the parent Dalvik process for all Dalvik VM instances in the system. The Zygote process also sets up a BSD read socket and listens for incoming requests. Once a new request is received, the Zygote process forks the parent Dalvik VM process to create a new Dalvik VM instance and then sends the child process to the requesting application.This way, the Dalvik create multiple instances of the runtime. [12]
The figure 2 represents diagramaticaly the android booting process and creation of the dalvik instances.

Dalvik VM is compactible with dalvik byte code (*.dex files).The *.dex files are created by the DEX compiler, which is part of the Android system (see Figure 3). This tool takes Java class files created by the Java compiler as the

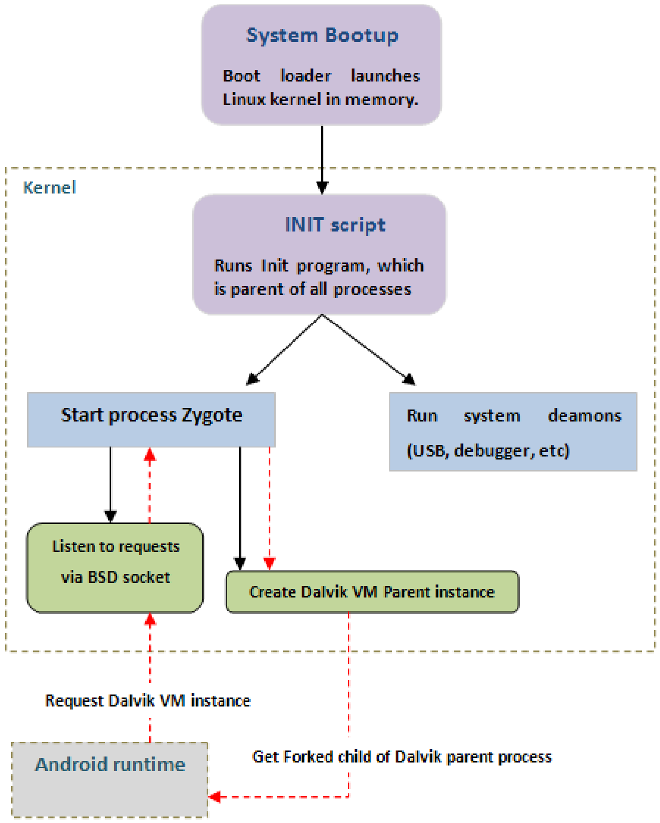

System Bootup
Boot loader launches Linux kernel in memory.
Kernel
INIT script
Runs Init program, which is parent of all processes
Start process Zygote
Run system deamons (USB, debugger, etc)
Listen to requests via BSD socket
Create Dalvik VM Parent instance
Request Dalvik VM instance
Android runtime
Get Forked child of Dalvik parent process

Fig. 2. Android Boot process

input.The DEX compiler then converts the .java file into a .dex file, which is of less size and more optimized for the Dalvik VM execution.

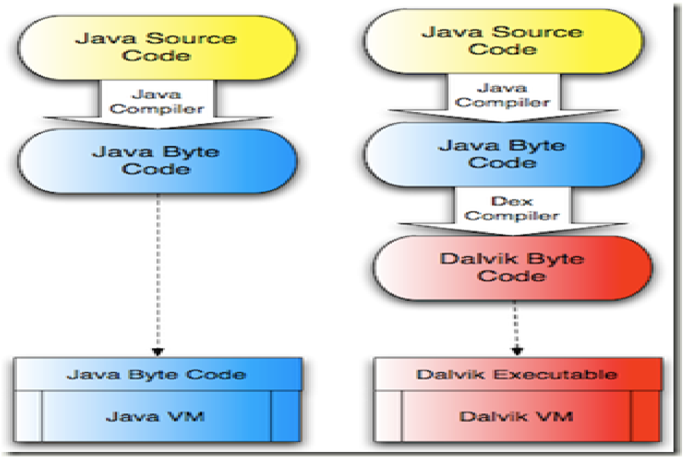


Fig. 3. *.dex creation

3.1.1 Dalvik-like register-based VM implementation

Below is a snippet of the code implementing the register-based virtual machine (Dalvik) in C.

```
/*
 * myvm.c
 */
#include "stdio.h"

#define NUM_REGS 4
#define TRUE 1
#define FALSE 0
#define INVALID -1

enum opCodes {
    HALT = 0x0,
    LOAD = 0x1,
    ADD  = 0x2,
};

/*
 * Register set of the VM
 */
int regs[NUM_REGS];

/*
 * VM specific data for an instruction
 */
struct VMData_ {
  int reg0;
  int reg1;
  int reg2;
  int reg3;
  int op;
  int scal;
};
typedef struct VMData_ VMData;

int fetch();
void decode(int instruction, VMData
    *data);
void execute(VMData *data);
void run();

 /*
  * Sample program with an instruction set
  */

unsigned int code[] = {0x1014,
                  0x110A,
                  0x2201,
                  0x0000};

/*
 * Instruction cycle: Fetch, Decode,
    Execute
 */
```

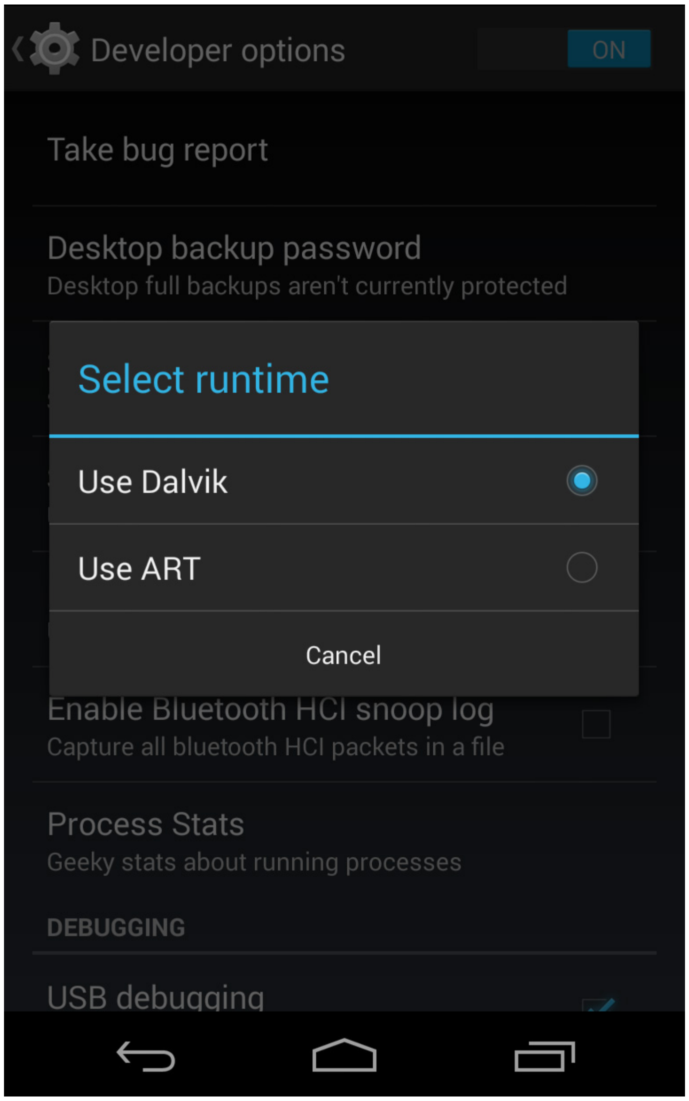
```
/*
 * Current state of machine: It's a
    binary true/false
 */

int running = TRUE;

/*
 * Fetch
 */
int fetch()
{
  /*
   * Program Counter
   */
  static int pc = 0;

  if (pc == NUM_REGS)
    return INVALID;

  return code[pc++];
}

void decode(int instr, VMData *t)
{
  t->op = (instr & 0xF000) >> 12;
  t->reg1 = (instr & 0x0F00) >> 8;
  t->reg2 = (instr & 0x00F0) >> 4;
  t->reg3 = (instr & 0x000F);
  t->scal = (instr & 0x00FF);
}

void execute(VMData *t)
{
  switch(t->op) {
    case 1:
      /* LOAD */
      printf("\nLOAD REG%d %d\n", t->reg1,
           t->scal);
      regs[t->reg1] = t->scal;
      break;

    case 2:
      /* ADD */
      printf("\nADD %d %d\n",
           regs[t->reg2], regs[t->reg3]);
      regs[t->reg1] = regs[t->reg2] +
           regs[t->reg3];
      printf("\nResult: %d\n",
           regs[t->reg1]);
      break;

    case 0:
    default:
      /* Halt the machine */
      printf("\nHalt!\n");
      running = FALSE;
      break;
  }
}

void run()
{
  int instr;
  VMData t;

  while(running)
  {
    instr = fetch();

    if (INVALID == instr)
      break;

    decode(instr, &t);
    execute(&t);
  }
}

int main()
{
  run();
  return 0;
}
```

4 Android Runtime -ART

ART stands for Android Runtime which is written in the native C/C++ langauge. ART uses Ahead-of-Time compilation(AOT).It pre-compiles a task and converts it into Low level Machine language which is the simplest form a machine understands during the first installation.[6]This explains why the first boot in ART takes alot of time as apps are converted to their native code.Currently using ART is an option in the android 4.4 gadgets.ART can be switched on an Android device through navigating to Settings on Developer options (See figure 4)

4.1 Operations of ART

Though the first boot can take at most 20 minutes to complete as ART converts existing Dalvik runtime (libdvm.so) to the ART compatible (libart.so), the launching of the applications is extremely fast. ART uses OAT files rather than the ODEX files used by Dalvik.


Developer options ON

Take bug report

Desktop backup password
Desktop full backups aren't currently protected

Select runtime

Use Dalvik [✓]

Use ART

Cancel

Enable Bluetooth HCI snoop log
Capture all bluetooth HCI packets in a file

Process Stats
Geeky stats about running processes

DEBUGGING

USB debugging

Fig. 4. ART selection

ART has two compiler backends,both using AOT Compilation method.One of them is enhanced through LLVM (Low Level Virtual Machine) architecture written in C++; all this designed for compile-time, link-time, run-time, and ”idle-time” optimization. The code snippet below shows a portion of the native code that converts .dex to .oat for ART compilation.[8]

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <valgrind.h>

#include <fstream>
#include <iostream>
#include <sstream>
#include <string>
#include <vector>

#include "base/stl_util.h"
#include "base/stringpiece.h"
#include "base/timing_logger.h"
#include "base/unix_file/fd_file.h"
#include "class_linker.h"
#include "dex_file-inl.h"
#include "driver/compiler_driver.h"
#include "elf_fixup.h"
#include "elf_stripper.h"
#include "gc/space/image_space.h"
#include "gc/space/space-inl.h"
#include "image_writer.h"
#include "leb128.h"
#include "mirror/art_method-inl.h"
#include "mirror/class-inl.h"
#include "mirror/class_loader.h"
#include "mirror/object-inl.h"
#include "mirror/object_array-inl.h"
#include "oat_writer.h"
#include "object_utils.h"
#include "os.h"
#include "runtime.h"
#include "ScopedLocalRef.h"
#include "scoped_thread_state_change.h"
#include "sirt_ref.h"
#include "vector_output_stream.h"
#include "well_known_classes.h"
#include "zip_archive.h"

namespace art {

static void UsageErrorV(const char* fmt,
    va_list ap) {
  std::string error;
  StringAppendV(&error, fmt, ap);
  LOG(ERROR) << error;
}

static void UsageError(const char* fmt,
    ...) {
  va_list ap;
  va_start(ap, fmt);
  UsageErrorV(fmt, ap);
  va_end(ap);
}


}

class Dex2Oat {
 public:
  static bool Create(Dex2Oat** p_dex2oat,
                Runtime::Options& options,
                CompilerBackend
```

```
                 compiler_backend,
             InstructionSet
                 instruction_set,
             size_t thread_count)
  SHARED_TRYLOCK_FUNCTION(true,
      Locks::mutator_lock_) {
 if (!CreateRuntime(options,
     instruction_set)) {
   *p_dex2oat = NULL;
   return false;
 }
 *p_dex2oat = new
     Dex2Oat(Runtime::Current(),
     compiler_backend, instruction_set,
     thread_count);
 return true;
}

~Dex2Oat() {
 delete runtime_;
 VLOG(compiler) << "dex2oat took " <<
     PrettyDuration(NanoTime() -
     start_ns_)
          << " (threads: " <<
              thread_count_ << ")";
}


// Reads the class names
    (java.lang.Object) and returns a set
    of descriptors (Ljava/lang/Object;)
CompilerDriver::DescriptorSet*
    ReadImageClassesFromFile(const char*
    image_classes_filename) {
  UniquePtr<std::ifstream>
      image_classes_file(new
      std::ifstream(image_classes_filename,
 std::ifstream::in));
  if (image_classes_file.get() == NULL) {
    LOG(ERROR) << "Failed to open image
        classes file " <<
        image_classes_filename;
    return NULL;
  }
  UniquePtr<CompilerDriver::DescriptorSet>
  result
  (ReadImageClasses
  (*image_classes_file.get()));
  image_classes_file->close();
  return result.release();
}

CompilerDriver::DescriptorSet*
    ReadImageClasses(std::istream&
    image_classes_stream) {
  UniquePtr<CompilerDriver::DescriptorSet>
      image_classes(new
    CompilerDriver::DescriptorSet);
  while (image_classes_stream.good()) {
    std::string dot;
    std::getline(image_classes_stream,
        dot);
    if (StartsWith(dot, "#") ||
        dot.empty()) {
      continue;
    }
    std::string
     descriptor
     (DotToDescriptor(dot.c_str()));
    image_classes->insert(descriptor);
  }
  return image_classes.release();
}
```

Through these optimization, estimates and some benchmarks indicate that the new run-time is already capable of reducing execution time to half for most applications.

5 Comparision ART with Dalvik VM

5.1 Speed and perfomance

AOT is capable of cutting execution time in half for most applications for it uses two compiler backends, both Ahead-of-Time. One compiler has LLVM (Low Level Virtual Machine) which is written in C/C++ designed for program optimization. When any program is being installed, it is first converted to it's native language for optimization.This explains why apps installation takes a longer time but once installed, the application's runtime is greatly reduced.

On the other hand, Dalvik is slow less compared with ART because it's a virtual machine. Every VM incurs an overhead because of the virtualization layer.The best VMs run code at 2 times slower w.r.t to native code. Dalvik is 4x slower than native code, although the gap is not noticeable in regular use, atleast for opening and using apps.It uses a Just-in-Time compiler.When an application is launched in Dalvik VM, it is first converted to it's native-this causes the time delay in the execution. In essense, every time you open an app, all the different parts of the smartphone responsible for making that app work have scramble to assemble the code for the app to make it work on your device.Once you close the app, all those parts relax.This is what makes the phone too slow while using Dalvik VM.

A benchmark test performed by Android police using Linpack application; Linpack app produces a score by performing a series of calculations on floating point numbers -this test checks the speed of your Android device and compare it to other Android devices. The scores done on ART and Dalvik enabled phones indicate that perfomance speed on ART are 10-14% higher than on Dalvik.

The Xda-developers also performed another benchmark test, to compare the throughput while performing a quick sort on an integer array. [11]The results were represented graphically as shown in the figure 5. From the graph,

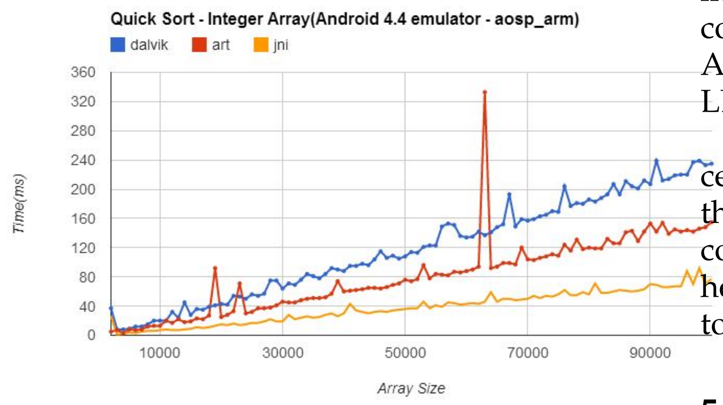


Fig. 5. Quick sort,integer array.-Dalvik vs ART vs JNI

JNI which uses pure native (C/C++) code takes the least time to perform a task, followed on the same is ART then Dalvik. The graph indicates that ART takes less time to finish a task less compared to Dalvik.

5.2 Memory Consumption

ART compiles all applications to machine level language during first installation and hence requires much more space for storage. It stores all the applications as pre-compiled code. To explain this: an apk can be termed as a zip file, ART extracts the 'zip' file to get the machine level code.The 'zip' is what dalvik uses to run directly but ART needs the extract (machine level code) before hand to run. This makes apps to using up more space than what dalvik uses.

Dalvik VM uses a memory cache- any app that is run is placed on the cache, once done with it is cleared and another one is placed. This saves more space while using Dalvik. ART works totally differently, it stores pre-complied apps in a more permanent fashion, thus more space is needed.

5.3 Processor type and capabilities

An advantage with Android over other platforms like iOS or Windows is that it runs on all sorts of hardware with more differentiation; i.e. arm v7, v7a, v8 x86, Intel etc and for different instruction sets. ART has been extensively optimized. The 64 bit android OS is compatible with the new range of ARM processors and it makes even better use of the lower-powered cores in ARMs big.LITTLE architecture.With ART a fewer cores will be used because of it's LLVM achitecture.

ART is be able to run on low end processors, optimizing the runtime, smoothening the graphics and enhancing the 'idle time'.Less compared to Dalvik, ART saves CPU cycles and hence saves battery because CPU doesnt need to recompile an app again and again.

5.4 Compactability

Since ART is a new Google project,it lacks compactability with a few android applications.Recently Android 4.4.2 fixed a number of incompatibility issues -some even with mega apps such as WhatsApp etc.Currently 99% of android applications are now compatible with ART. Any application which is not compactible with ART requires a switchback to the old Dalvik runtime machine.

6 Views and Opinions

Google ART will be a key solution to the speed and graphics optimization in the Android OS. Since its written in the native C/C++ language, it will alter the normal use of the DEX compiler on the Dalvik runtime. Both the Android developers and users will benefit as there will be comparison in performance across platforms ie. an Iphone with 1 GB RAM will be the same as a Samsung S4 with 2 GB RAM.

ART will enhance optimization on the android OS through several ways such us;

1) Much speed and increased throughput
   Dalvik being a Just-in-time (JIT) process VM has to transform app source code into an executable program every time it starts. This creates a time lag during the starting of an application.
   By contrast ART uses the Ahead-of-time (AOT) process and thus compiles when an app is first installed. Using this method apps can launch up to twice as quickly. This dramatically enhances the speed of the android phone.
2) Battery life improvement
   AOT processing makes app-launch a less intensive process, so your phone doesnt have to work as hard to compile an application each time its launched. This increases battery life. This is noted when you have background processes running on your phone while using ART.[10]
   Early developer reports claim to see as much as a 30% boost in idle standby time, but there are so many variables here, thus becomes hard to measure.
3) Smoothened Graphics
   Applications will benefit from smoother animations and more instantaneous responses to touch and other sensor data.This will enhance gamming applications in android OS.This is an advatage to both users and developers.
4) Universal compactability ART being part the android OS, fits in a range of hardware devices with different processors. AOT frees the developer from having to compile for all current and future architecture, while precompiling would require very fat binaries. ART does the compilation of any application to machine code thus removing the need of having an enhanced IDE (Intergrated Development Environment). Any language with an LLVM front end could be used to develop apps with ART thus no language limitation. Devices with a fewer processor cores will perform with optimized speed.
5) ART will have JNI capabilities
   JNI is a programing framework that enables java code running on Java Virtual Machine (JVM) to call and be called by native applications on certain hardware resources. This will increase optimization in ART while running native applications.

Despite Android OS optimization ART contains a few downfalls;

1) It requires more storage.
   Installed apps will use more storage roughly 10% to 20% more. This is because the pre-compiled apps which are stored in their native version on the phone. Dalvik has a cache for loading applications on-demand method.
2) Some Android apps are incompatible with ART
   A few android apps are not compactible with ART ie. Whatsup application. Such applications call for a switch back to the Dalvik runtime. This is a challenge to developers who are limited to different architectures only.
3) Long time on Booting.
   An Android phone using ART may take atmost 30 minutes during startup.This is a serious test of patience.This is because it compiles all the applications in the device into their native forms. ART stores applications in their compiled native form.
4) ART is available only one Android Version
   ART is only found on Android 4.4. This is as well on expensive Android devices on the market.The proliferation of this devices in the market is thus low.

7 Related Work

There is a new Android OS called CyanogenMod OS created by Oneplus inc. [14]The Android 4.4.2 based OS is made for the device called ”One”. The company has tweaked the Android OS adding in a Snapdragon 801 processor to enhance its power and thus android optimization. The phone, ’One’ was recently launched on 23RD April 2014.[15]

8 Conclusion and Future work

Through improvements ART will have a few undectable draw backs and reduce the lag in the android devices.From some benchmark study, the runtime will enhance the battery life. ART supports the JNI codes, this will lead to further optimization of the applications.

ART spurs the rise to more technologies in Google including the Google wearables, NFC, IR Blaster and Bluetooth Additions.This is enhanced due to dramatic speed of execution experienced on the Android devices.

With the ability to run JNI applications, we need to do some research in the coming future to compare and benchmark the performance of Android applications running on ART and JNI android applications running of ART. This can advice on the direction to which Google should take to enhance further optimization on their gadgets.

It's eminent that Ahead-of-Time (AOT) is the future compilation process, in which Google has adopted. In view of the trends, ART might be the standard runtime for Android 5.0.

References

[1] (Introducing ART)- *http://source.android.com/devices/tech/dalvik/art.html* Retrieved on 8th April 2014

[2] (The ART of Android) Xda-developers *http://www.xda-developers.com/android/the-art-of-android/* Retrieved on 9th April 2014

[3] (Android Preparing to Become a 64-bit OS, ART Will Replace Dalvik as Default Runtime Compiler) Xda-developers *http://www.xda-developers.com/android/android-preparing-to-become-a-64-bit-os-art-will-replace-dalvik-as-default-runtime-compiler* Retrieved on 9th April 2014

[4] Experimental Google ART in android Kitkat can bring twice faster app executions *http://www.phonearena.com/news/Experimental-Google-ART-runtime-in-Android-KitKat-can-bring-twice-faster-app-executions*

[5] (ART vs Dalvik) - introducing the new Android runtime in KitKat *https://www.infinum.co/the-capsized-eight/articles/art-vs-dalvik-introducing-the-new-android-runtime-in-kit-kat* Retrieved on 9th April 2014

[6] New Runtime Compiler in Android 4.4 to Possibly Bring Better Performance in Future Releases *http://www.xda-developers.com/android/new-runtime-compiler-in-android-4-4/* Retrieved on 9th April 2014

[7] Google acquires France Fexycore for better performance *http://gigaom.com/2013/10/22/google-buys-frances-flexycore-for-better-performing-android/* Retrieved on 10th April 2014

[8] dex2oat file which converts Odex files into oat files compatible with ART *https://android.googlesource.com/platform/art/+/kitkat-release/dex2oat/dex2oat.cc*

[9] Add DEXPREOPT support for ART *https://github.com/olibc/build/commit/ced4bff58e76a16ebce3a35ed24aadc8490* Retrieved on 9th April 2014

[10] Android Battery life *http://lifehacker.com/android-art-vs-dalvik-runtimes-effect-on-battery-life-1507264545* Retrieved on 16th April 2014

[11] ART RUNTIME FOR ANDROID EXPLAINED *http://sourcex.wordpress.com/2013/11/12/art-runtime-for-kitkat-4-4-explained/* Retrieved on 9th April 2014

[12] Dalvik internals *https://sites.google.com/site/io/dalvik-vm-internals* Retrieved on 9th April 2014

[13] Android ART *http://it.laplit.com/index.php/humanoid-blog/144-android-os-art* Retrieved on 9th April 2014

[14] OnePlus One is a powerhouse Android smartphone running CyanogenMod, starts from $299 *http://thenextweb.com/gadgets/2014/04/23/oneplus-unveils-one-powerhouse-android-smartphone-cyanogenmod-starting-299* Retrieved on 23rd April 2014

[15] Oneplus One Phone *http://eplus.net/one- one phone* Retrieved on 23rd April 2014